

FIG. 1 is a block diagram of a network system.

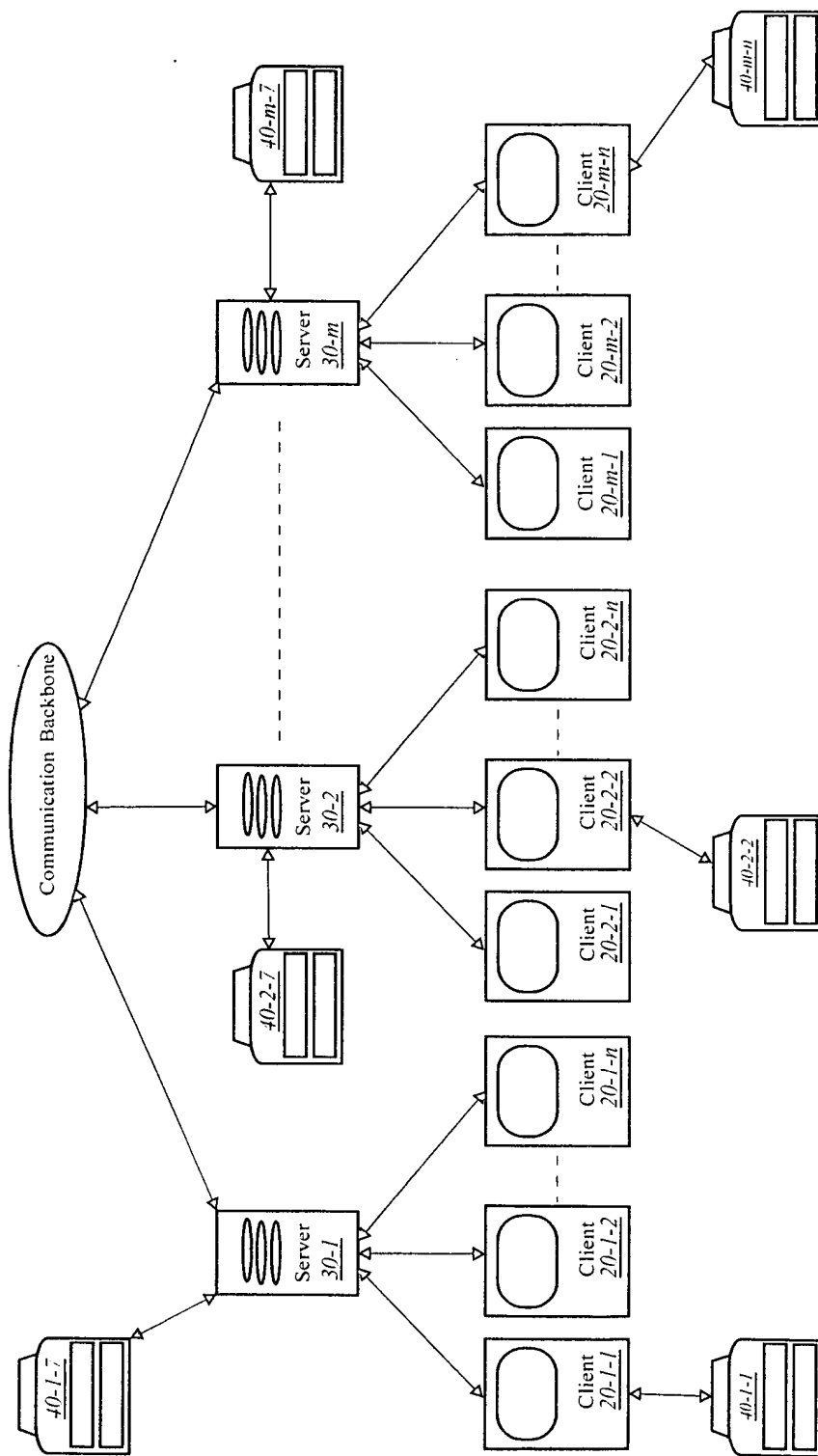


Fig. 1

Client Flow Chart

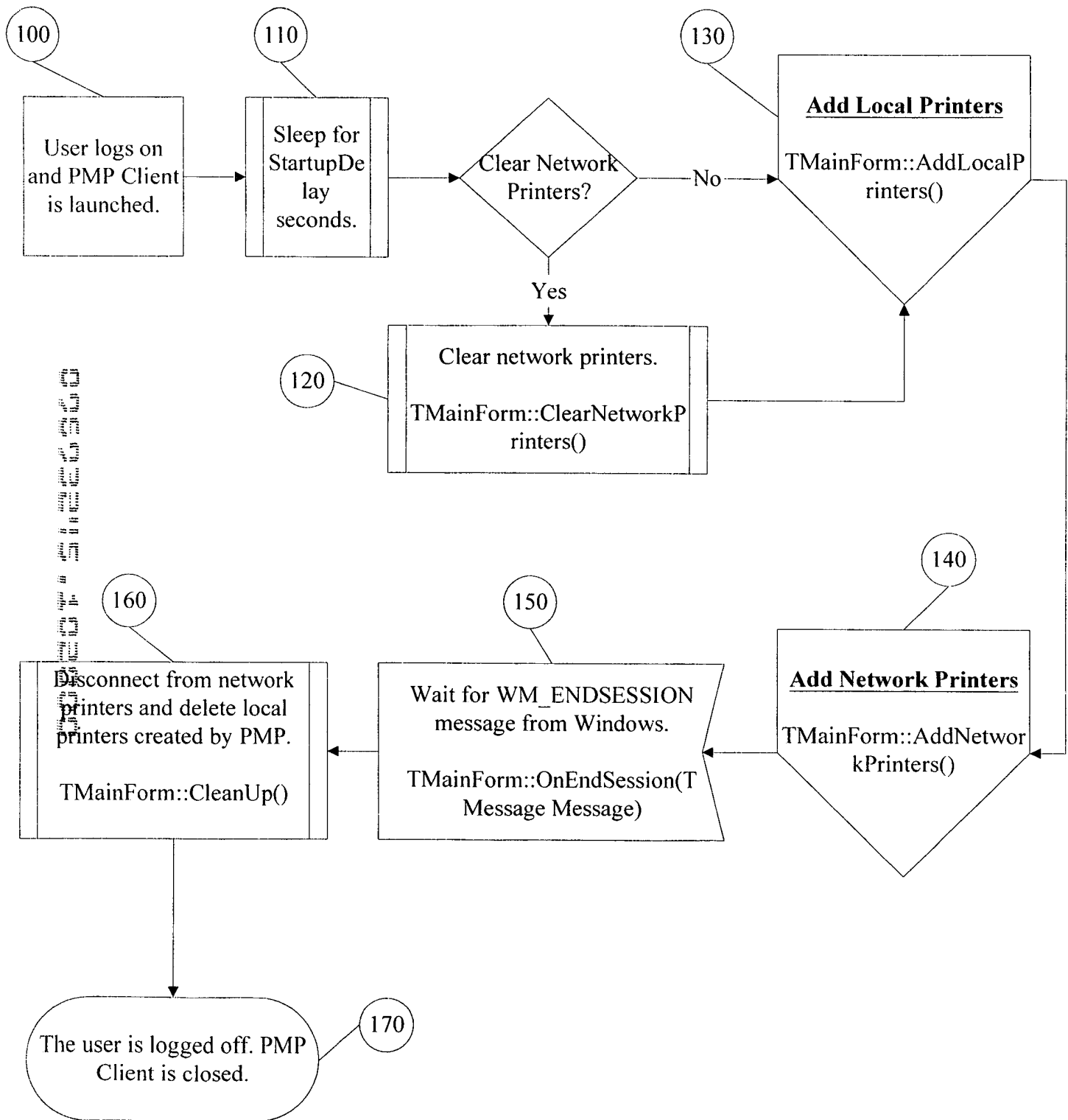


Fig. 2

Add Local Printers

MainForm::AddLocalPrinters()

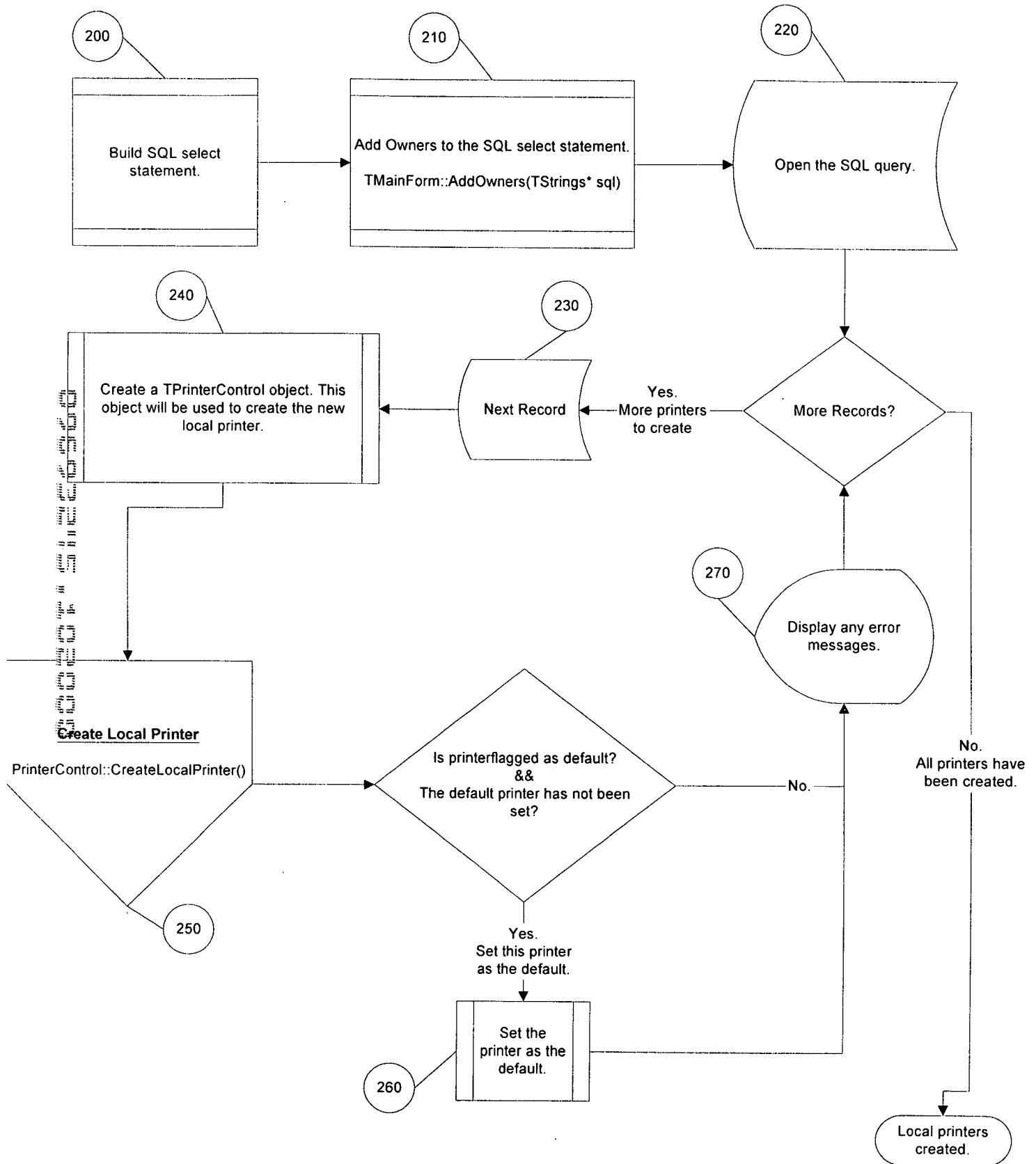


Fig. 3

Create Local Printer

PrinterControl::CreateLocalPrinter()

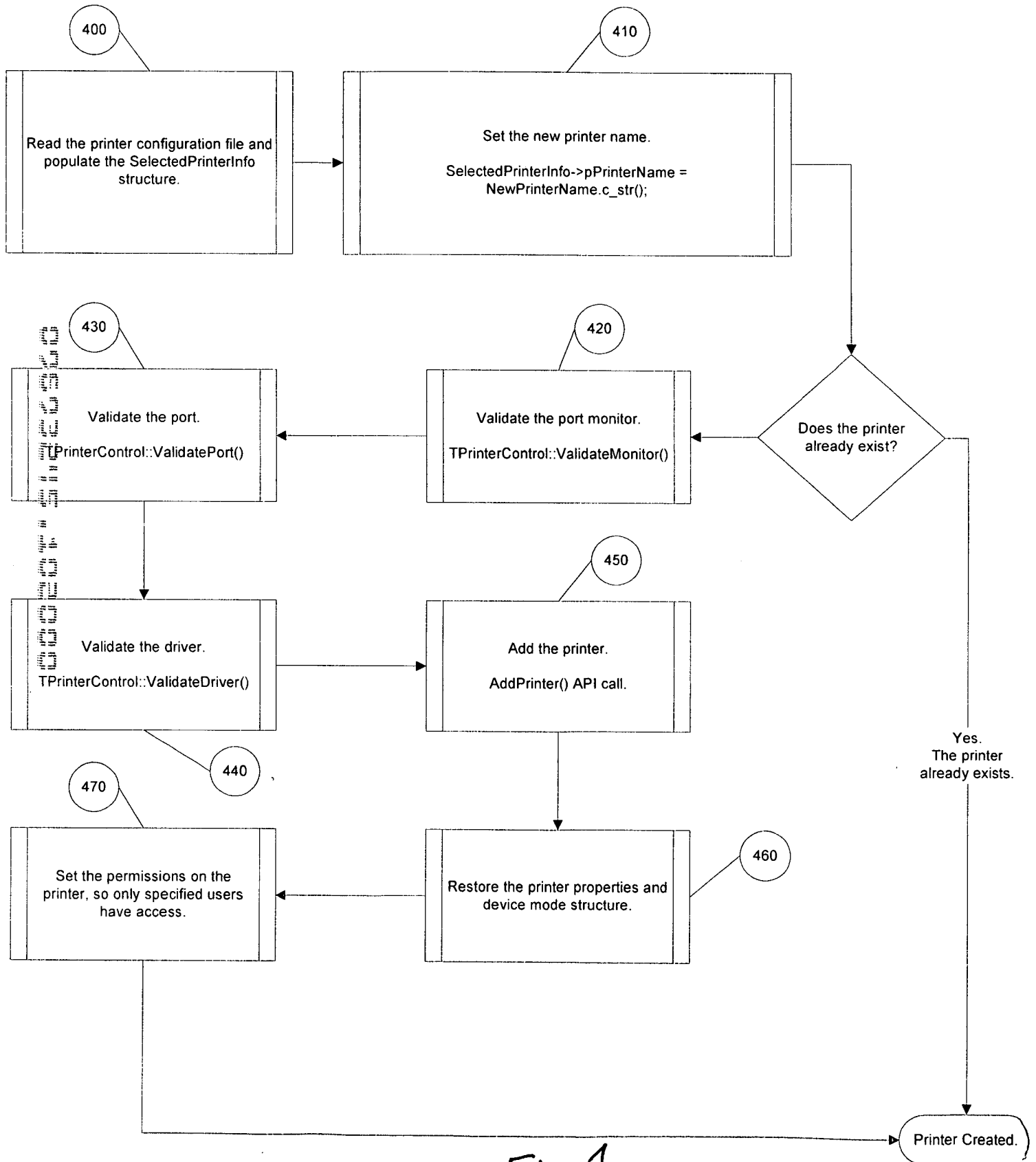


Fig. 4

Add Network Printers

MainForm::AddNetworkPrinters()

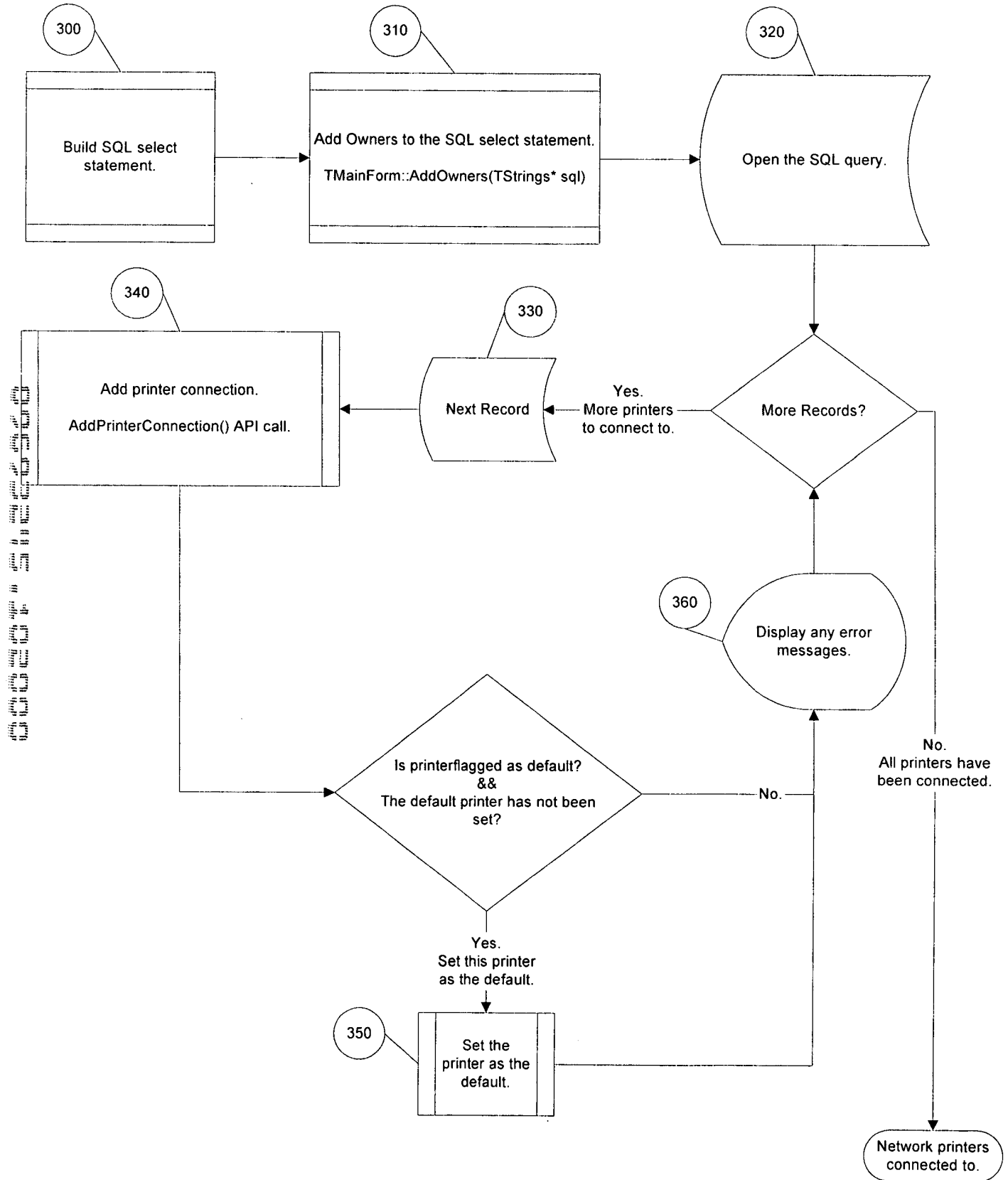


Fig. 5

Implementation CODE

MainUnit.h

```
//-----
#ifndef MainUnitH
#define MainUnitH
//-----

#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Dbtables.hpp>
#include <NetworkInfo.h>
#include <ShellApi.h>
#include <ExtCtrls.hpp>
#include <TriceratMessaging.h>
#include <DirTools.h>

//-----

class TMainForm : public TForm
{
__published:    // IDE-managed Components
    TNetworkInfo *FNetworkInfo;
    TButton *CloseBtn;
    TTimer *IcaPrinterSecurity;
    TTimer *Initialize;
    void __fastcall CloseBtnClick(TObject *Sender);
    void __fastcall FormCreate(TObject *Sender);
    void __fastcall FormShow(TObject *Sender);
    void __fastcall InitializeTimer(TObject *Sender);
    void __fastcall FormHide(TObject *Sender);
    void __fastcall FormActivate(TObject *Sender);
    void __fastcall IcaPrinterSecurityTimer(TObject *Sender);
    void __fastcall FormClose(TObject *Sender, TCloseAction &Action);

private: // User declarations
    AnsiString PrinterInfoPath;
    TStringList *LocalPrinters;
    TStringList *NetworkPrinters;
    bool bClearNetworkPrinters;
    bool bSetIcaPrinterRights;
    bool DefaultPrinterSet;
    bool Initializing;
    int IcaPrinterRightsDelay;
    int StartupDelay;

    void __fastcall AddOwners(TStrings* sql);
    void __fastcall AddLocalPrinters();
    void __fastcall AddNetworkPrinters();
    void OnDesktopInit(TMessage Message);
    void ClearNetworkPrinters();
    void CleanUp();
    void OnQueryEndSession(TMessage Message);
    void OnEndSession(TMessage Message);
    bool GetPrinterRights(TStringList * Users);
```

FIG. 6.1

```

public:          // User declarati
__fastcall TMainForm(TComponent* Owner);
__fastcall ~TMainForm();
int ProductID;
AnsiString LogFile;
TDirTools *DirTools;

protected:
BEGIN_MESSAGE_MAP
    VCL_MESSAGE_HANDLER(TM_D2K_INIT, TMessage, OnDesktopInit)
    VCL_MESSAGE_HANDLER(WM_ENDSESSION, TMessage, OnEndSession)
END_MESSAGE_MAP(TForm)
};
//-----
extern PACKAGE TMainForm *MainForm;
//-----
#endif

```

```

MainUnit.Cpp
#include <vcl.h>
#pragma hdrstop
//-----
#include "MainUnit.h"
#include "NetworkInfo.h"
#include <PrinterControl.h>
#include <RegTools.h>
//-----
#pragma package(smart_init)
#pragma link "NetworkInfo"
#pragma resource "*.dfm"
//-----
TMainForm *MainForm;
//-----

```

STEP 100

```

__fastcall TMainForm::TMainForm(TComponent* Owner)
: TForm(Owner)
{
    Session->Active = false;
    LocalPrinters = new TStringList;
    NetworkPrinters = new TStringList;
    DefaultPrinterSet = false;
    Initializing = false;
}
//-----

```

STEP 170

```

__fastcall TMainForm::~TMainForm()
{

```

FIG. 6.2

```

delete LocalPrinters;
delete NetworkPrinters;
DirTools->WriteLog(LogFile, "Terminating PMP Client");
delete DirTools;
}
//-----

void __fastcall TMainForm::AddOwners(TStrings* sql)
{
    AnsiString ClientName;
    AnsiString ComputerName;
    ClientName = getenv("CLIENTNAME");
    if (!ClientName.IsEmpty())
        ClientName = ClientName.UpperCase();
    ComputerName = getenv("COMPUTERNAME");

    FNetworkInfo->Clear();

    sql->Add(" IN (SELECT ID FROM Owners WHERE Name = '" +
        FNetworkInfo->UserName + "'");

    if (FNetworkInfo->LocalComputerName != ("\\\\\\" + FNetworkInfo->DomainName))
    {
        FNetworkInfo->SourceServerName = FNetworkInfo->DomainControllerName;

        for (int i = 0; i < FNetworkInfo->MyGlobalGroupCount; i++)
            sql->Add(" OR Name = '" + FNetworkInfo->MyGlobalGroupNames[i] + "'");
    }

    FNetworkInfo->SourceServerName = "";

    for (int i = 0; i < FNetworkInfo->MyLocalGroupCount; i++)
        sql->Add(" OR Name = '" + FNetworkInfo->MyLocalGroupNames[i] + "'");

    if (!ClientName.IsEmpty() && ClientName != ComputerName)
        sql->Add(" OR Name = '" + ClientName + "'");
    if (!ComputerName.IsEmpty())
        sql->Add(" OR Name = '" + ComputerName + "'");

    sql->Add(")");
}
//-----

```

STEP 130

```

void __fastcall TMainForm::AddLocalPrinters()
{
    TQuery* query = new TQuery(NULL);
    int i;
    AnsiString SourceServer;
    AnsiString Monitor;
    AnsiString Port;
    AnsiString FileName;
    AnsiString PrinterName;
    AnsiString NewPrinterName;
    AnsiString ClientName;
    bool IsDefault;

```

FIG. 6.3


```
TStringList *Messages = new TStringList();
TStringList *Users = new TStringList();
```

```
GetPrinterRights(Users);
```

```
ClientName = getenv("CLIENTNAME");
if (!ClientName.IsEmpty())
    ClientName = ClientName.UpperCase();
else
    ClientName = FNetworkInfo->UserName;
```

```
query->DatabaseName = "Tricerat PMP";
```

STEP 200

```
query->SQL->Add("SELECT o.Ordinal, a.Ordinal, p.FileName, p.Name, ");
query->SQL->Add("p.Port, p.Monitor, p.SourceServer, a.IsDefault ");
query->SQL->Add("FROM Owners o, AssignedLocalPrinters a, LocalPrinters p ");
query->SQL->Add("WHERE o.ID = a.OwnerID AND a.LocalPrinterID = p.ID ");
query->SQL->Add("AND p.Disabled = False ");
query->SQL->Add("AND a.OwnerID ");
```

STEP 210

```
AddOwners(query->SQL);

query->SQL->Add(" ORDER BY Ordinal");
```

```
try
{
```

STEP 220

```
query->Open();
```

STEP 230

```
i = -1;
while (query->Active && !query->Eof && query->RecordCount > ++i)
{
    //Add printers here.
    SourceServer = query->FieldByName("SourceServer")->AsString;
    Monitor = query->FieldByName("Monitor")->AsString;
    Port = query->FieldByName("Port")->AsString;
    FileName = query->FieldByName("FileName")->AsString;
    PrinterName = query->FieldByName("Name")->AsString;
    IsDefault = query->FieldByName("IsDefault")->AsBoolean;
    NewPrinterName = ClientName + "#" + PrinterName;
```

```
try
{
```

STEP 240

```
//Constructor to point to local computer for drivers.
TPrinterControl *PrinterControl = new TPrinterControl(
    PrinterInfoPath, SourceServer);
```

```
if (!Port.IsEmpty() && !Monitor.IsEmpty())
    PrinterControl->RemapPort(Port, Monitor);
```

STEP 250

```
//Create the temp printer.
if (PrinterControl->CreateLocalPrinter(FileName, NewPrinterName, Users))
```

FIG. 6.4

```

    {
        LocalPrinters->Add(NewPrinterName);
        if (IsDefault && !DefaultPrinterSet)
        {
STEP 260            if (PrinterControl->SetDefaultPrinter(NewPrinterName))
                    DefaultPrinterSet = true;
        }
    }

    if (0 < PrinterControl->Messages->Count)
        Messages->Add(PrinterControl->Messages->Text);

    delete PrinterControl;
}
catch(...)
{
    Messages->Add("Error Creating Printer \"" + NewPrinterName + "\"");
}

query->FindNext();

    Next();
}
}
catch (...)
{
}

query->Close();
delete query;

Users->Clear();
delete Users;
STEP 270
if (0 < Messages->Count)
{
    MessageBox(NULL, Messages->Text.c_str(), "PMP CLient",
        MB_OK | MB_ICONERROR | MB_SYSTEMMODAL);
}
}
//-----

STEP 140
void __fastcall TMainForm::AddNetworkPrinters()
{
    TQuery* query = new TQuery(NULL);
    int i;
    AnsiString Map;
    AnsiString PrinterName;
    AnsiString FullShareName;
    AnsiString FullPrinterName;
    AnsiString Argument;
    bool IsDefault;
    DWORD dwError;

```

FIG. 6.5

```
query->DatabaseName = "TricorePMP";
```

STEP 300

```
query->SQL->Add("SELECT o.Ordinal, a.Ordinal, p.Name, a.Map, a.IsDefault ");
query->SQL->Add("FROM Owners o, AssignedNetworkPrinters a, NetworkPrinters p ");
query->SQL->Add("WHERE o.ID = a.OwnerID AND a.NetworkPrinterID = p.ID ");
query->SQL->Add("AND p.Disabled = False ");
query->SQL->Add("AND a.OwnerID ");
```

STEP 310

```
AddOwners(query->SQL);

query->SQL->Add(" ORDER BY Ordinal");
```

```
try
{
    //Constructor to point to local computer for drivers.
    TPrinterControl *PrinterControl = new TPrinterControl(
        NULL, NULL);
```

STEP 320

```
query->Open();
```

STEP 330

```
i = -1;
while (query->Active && !query->Eof && query->RecordCount > ++i)
{
    //Add printers here.
    PrinterName = query->FieldByName("Name")->AsString;
    Map = query->FieldByName("Map")->AsString;
    IsDefault = query->FieldByName("IsDefault")->AsBoolean;
```

STEP 340

```
if (!AddPrinterConnection(PrinterName.c_str()))
{
    dwError = GetLastError();
    AnsiString Message;
    Message = "Unable to connect to printer " + PrinterName + " \n\n";
    Message = Message + "Error Code = " + String(dwError);

    query->FindNext();
    continue;
}

FullShareName = PrinterControl->GetPrinterShareName(PrinterName);
FullPrinterName = PrinterControl->GetPrinterFullName(PrinterName);

NetworkPrinters->Add(FullPrinterName);
```

STEP 350

```
if (IsDefault && !DefaultPrinterSet)
{
    if (!PrinterControl->SetDefaultPrinter(FullPrinterName))
        ShowMessage(PrinterControl->Messages->Text);

    DefaultPrinterSet = true;
}
```

FIG. 6.6

```

if (!Map.IsEmpty())
{
    Argument = "use " + Map + " /d";
    ShellExecute(NULL, "open", "net", Argument.c_str(),
        NULL, SW_HIDE);

    Argument = "use " + Map + " " + FullShareName;
    ShellExecute(NULL, "open", "net", Argument.c_str(),
        NULL, SW_HIDE);
}

```

STEP 360

```

MessageBox(NULL, Message.c_str(), "PMPClient",
    MB_OK | MB_ICONERROR | MB_SYSTEMMODAL);

query->FindNext();

Next();
}

```

```

    delete PrinterControl;
}
catch (...)
{
}

query->Close();
delete query;
}
//-----

void __fastcall TMainForm::CloseBtnClick(TObject *Sender)
{
    CleanUp();
}
//-----

```

```

void __fastcall TMainForm::FormCreate(TObject *Sender)
{
    FormHide(Sender);
}
//-----

```

```

void __fastcall TMainForm::FormShow(TObject *Sender)
{
    TRegistry *Reg = new TRegistry;
    LogFile = String(getenv("TEMP")) + "\\PMP.txt";
    DirTools = new TDirTools();

    ShowWindow(Application->Handle, SW_HIDE);

    Reg->RootKey = HKEY_LOCAL_MACHINE;
}

```

FIG. 6.7

```

if (Reg->OpenKey("Software\\Microsoft\\Printer\\PMP", true))
{
    PrinterInfoPath = Reg->ReadString("PrinterInfo Path");

    try
    {
        bClearNetworkPrinters = Reg->ReadBool("ClearNetworkPrinters");
    }
    catch (...)
    {
        bClearNetworkPrinters = false;
        Reg->WriteBool("ClearNetworkPrinters", bClearNetworkPrinters);
    }

    try
    {
        bSetIcaPrinterRights = Reg->ReadBool("SetIcaPrinterRights");
    }
    catch (...)
    {
        bSetIcaPrinterRights = false;
        Reg->WriteBool("SetIcaPrinterRights", bSetIcaPrinterRights);
    }

    try
    {
        IcaPrinterRightsDelay = Reg->ReadInteger("IcaPrinterRightsDelay");
    }
    catch (...)
    {
        IcaPrinterRightsDelay = 15;
        Reg->WriteInteger("IcaPrinterRightsDelay", IcaPrinterRightsDelay);
    }

    try
    {
        StartupDelay = Reg->ReadInteger("StartupDelay");
    }
    catch (...)
    {
        StartupDelay = 30;
        Reg->WriteInteger("StartupDelay", StartupDelay);
    }

}
Reg->CloseKey();
Reg->Free();

if (PrinterInfoPath.IsEmpty())
{
    MessageBox(NULL, "Unable to Read Registry Values!", "PMPClient",
        MB_OK | MB_ICONERROR | MB_SYSTEMMODAL);
    Close();
}

if (5 < StartupDelay)

```

FIG. 6.8

```

Initialize->Interval = StartupDelay * 1000;
else
    Initialize->Interval = 5000;

DirTools->WriteLog(LogFile, "StartupDelay = " + String(Initialize->Interval));

```

STEP 110

//This can be stopped if Desktop sends us a message.

```
Initialize->Enabled = true;
```

```

if (bSetIcaPrinterRights)
{
    if (5 < IcaPrinterRightsDelay)
        IcaPrinterSecurity->Interval = IcaPrinterRightsDelay * 1000;
    else
        IcaPrinterSecurity->Interval = 5000;

    IcaPrinterSecurity->Enabled = true;
}
}
//-----

```

STEP 160

```

void TMainForm::CleanUp()
{
    int i;
    HWND hWnd;

    //Wait for RegSet.
    hWnd = (HWND)1;
    while (NULL != hWnd)
    {
        hWnd = FindWindow("TRegSetMainForm", NULL);

        if (NULL != hWnd)
        {
            SendMessage(hWnd, WM_CLOSE, NULL, NULL);
        }

        Sleep(100);
    }

    try
    {
        //Constructor to point to local computer for drivers.
        TPrinterControl *PrinterControl = new TPrinterControl(
            NULL, NULL);

        i = -1;
        while (LocalPrinters->Count > ++i)
        {
            PrinterControl->DeleteLocalPrinter(LocalPrinters->Strings[i]);
        }

        delete PrinterControl;

        i = -1;
    }
}

```

FIG. 6.9

```

while (NetworkPrinters->Count < ++i)
{
    DeletePrinterConnection(NetworkPrinters->Strings[i].c_str());
}
}
catch(...)
{
}
}
//-----

```

STEP 120

```
void TMainForm::ClearNetworkPrinters()
```

```

{
    try
    {
        //Constructor to point to local computer for drivers.
        TPrinterControl *PrinterControl = new TPrinterControl(
            NULL, NULL);

        PrinterControl->ClearNetworkPrinters();

        delete PrinterControl;
    }
    catch(...)
    {
    }
}
//-----

```

```
void __fastcall TMainForm::InitializeTimer(TObject *Sender)
```

```

{
    Initialize->Enabled = false;
    Initializing = true;
    LogFile = String(getenv("TEMP")) + "\\PMP.txt";
    DirTools = new TDirTools();

    try
    {
        if (bClearNetworkPrinters)
        {
            DirTools->WriteLog(LogFile, "Clearing Network Printers");
            ClearNetworkPrinters();
        }

        Session->Active = true;

        DirTools->WriteLog(LogFile, "Add Local Printers");
        AddLocalPrinters();
        DirTools->WriteLog(LogFile, "Finished With Local Printers");

        DirTools->WriteLog(LogFile, "Add Network Printers");
        AddNetworkPrinters();
        DirTools->WriteLog(LogFile, "Finished With Network Printers");

        Session->Active = false;
    }
    catch(...)
    {
    }
}

```

FIG. 6.10

```

    }
    catch(...)
    {
    }

    Initializing = false;
}
//-----

void __fastcall TMainForm::IcaPrinterSecurityTimer(TObject *Sender)
{
    IcaPrinterSecurity->Enabled = false;

    try
    {
        //Constructor to point to local computer for drivers.
        TPrinterControl *PrinterControl = new TPrinterControl(
            NULL, NULL);

        PrinterControl->SetIcaPrinterRights();

        delete PrinterControl;
    }
    catch(...)
    {
    }
}
//-----

void __fastcall TMainForm::FormHide(TObject *Sender)
{
    ShowWindow(Application->Handle, SW_HIDE);
    BorderStyle = bsNone;
    Width = 0;
    Height = 0;
}
//-----

void __fastcall TMainForm::FormActivate(TObject *Sender)
{
    ShowWindow(Application->Handle, SW_HIDE);
}
//-----

void TMainForm::OnDesktopInit(TMessage Message)
{
    if (0 == Message.WParam)
    {
        DirTools->WriteLog(LogFile, "PMP Received Message Desktop is Initializing");
        while(Initializing)
        {
            Sleep(1000);
        }
    }
}

```

FIG. 6.11


```

    }

    Initialize->Enabled = false;
}

if (1 == Message.WParam)
{
    DirTools->WriteLog(LogFile, "PMP Received Message From Desktop to Initialize");
    DefaultPrinterSet = false;
    Initialize->Enabled = false;
    Initialize->Interval = 1000;
    Initialize->Enabled = true;
}
}
//-----

```

STEP 150

```
void TMainForm::OnEndSession(TMessage Message)
```

```

{
    DirTools->WriteLog(LogFile, "PMP Cleanup In Progress");
    CleanUp();
    DirTools->WriteLog(LogFile, "PMP Cleanup Finished");
    Application->Terminate();
}
//-----

```

```
void __fastcall TMainForm::FormClose(TObject *Sender, TCloseAction &Action)
```

```

{
    CleanUp();
}
//-----

```

```
bool TMainForm::GetPrinterRights(TStringList * Users)
```

```

{
    TRegistry *Reg = new TRegistry();

    if (!Users)
        Users = new TStringList();

    Users->Clear();

    Reg->RootKey = HKEY_LOCAL_MACHINE;
    if (Reg->OpenKey("Software\\Tricerat\\PMP", true))
    {
        if (Reg->ValueExists("PrinterRights"))
        {
            try
            {
                AnsiString tempString;
                BYTE *pTemp = NULL;
                DWORD dwType = 0;
                DWORD dwSize = 0;
                int i = 0;

                RegQueryValueEx(Reg->CurrentKey, "PrinterRights",

```

FIG. 6.12

```

        NULL, &dwType, pTemp, &dwSize);

pTemp = (BYTE*)malloc(dwSize);
ZeroMemory(pTemp, dwSize);

RegQueryValueEx(Reg->CurrentKey, "PrinterRights",
    NULL, &dwType, pTemp, &dwSize);

if (0 < dwSize)
{
    i = -1;
    while ((int)dwSize > ++i)
    {
        if ('\0' == (char)pTemp[i])
        {
            if (!tempString.IsEmpty())
                Users->Add(tempString);

            tempString = "";
        }
        else
        {
            tempString = tempString + (char)pTemp[i];
        }
    }

    free(pTemp);
}
catch(...)
{
}
else
{
    RegSetValueEx(Reg->CurrentKey, "PrinterRights",
        NULL, REG_MULTI_SZ, NULL, 0);
}
}
Reg->CloseKey();
Reg->Free();

return true;
}

```

PrinterControl.h

```

//-----
#ifndef PrinterControlH
#define PrinterControlH
//-----
#include <SysUtils.hpp>
#include <Controls.hpp>
#include <Classes.hpp>
#include <Forms.hpp>
#include <winpool.h>

```

FIG. 6.13

```

#include <stdio.h>
#include <iostream.h>
#include <fstream.h>
#include <StUtils.hpp>
#include <RegTools.h>
#include "..\\DDK\\Inc\\winsplp.h"

#define CONTROL_FULL 1
#define TEMP_BUFFER_SIZE 128000

//-----
class PACKAGE TPrinterControl : public TComponent
{
private:
    static AnsiString CleanupFilename(AnsiString Filename);

protected:

    PRINTER_INFO_2 *SelectedPrinterInfo;
    DWORD SelectedPrinterInfoSize;
    AnsiString PrtInfoPath;
    AnsiString PrinterName;
    AnsiString PortMonitorDescription;
    AnsiString NewPrinterName;
    AnsiString SourceServerName;
    AnsiString NewPortName;
    AnsiString NewPortMonitor;
    DWORD dwDevModeSize;

    DRIVER_INFO_3 *GetRemoteDriverInfo(AnsiString ServerName,
        AnsiString DriverName);
    TStringList *CopyDriverFiles(TStringList *SourceFiles);
    bool ValidateDriver(AnsiString DriverName);
    bool ValidatePort(AnsiString PortName, AnsiString PortMonitor);
    bool ValidateMonitor(AnsiString MonitorName);
    bool PrinterSetOwnerOnlyRights(AnsiString PrinterName);
    bool PrinterSetCurrentUserOnlyRights(AnsiString PrinterName);
    bool PrinterAddAccessRights(AnsiString PrinterName, AnsiString UserName, int nAccess);
    bool WritePrinterInfo(AnsiString FileToSaveTo);
    bool ReadPrinterInfo(AnsiString FileToReadFrom);
    bool SaveLocalPrinter();
    bool CreateLocalPrinter();
    bool SetDefaultPrinter();
    AnsiString GetIcaClientPort(AnsiString OldPort);
    AnsiString GetPortMonitor(AnsiString PortName);

public:
    __fastcall TPrinterControl(AnsiString PathToPrinterInfoFiles,
        AnsiString SourceServerNameForDrivers);
    __fastcall ~TPrinterControl();

    bool PrinterAddAccessRights(AnsiString PrinterName, TStringList *Users, int nAccess);
    bool SetDefaultPrinter(AnsiString PrinterToSetAsDefault);
    bool CreateLocalPrinter(AnsiString PrinterToCreate);
    bool CreateLocalPrinter(AnsiString PrinterToCreate,

```

FIG. 6.14

```

        AnsiString NewPrinterToCreate);
bool CreateLocalPrinter(AnsiString PrinterToCreate,
        AnsiString NewPrinterToCreate, TStringList *Users);
bool SaveLocalPrinter(AnsiString PrinterToSave, AnsiString SaveName);
bool SaveLocalPrinter(AnsiString PrinterToSave);
bool RemapPort(AnsiString Port, AnsiString Monitor);
bool PrinterPropertiesDialog(AnsiString PrinterName, HANDLE hWnd);
bool DeleteLocalPrinter(AnsiString PrinterName);
static PRINTER_INFO_2 *GetPrinterInfo2(AnsiString PrinterName);
AnsiString GetStatusString(DWORD dwStatus);
TStringList *GetLocalDrivers();
TStringList *GetLocalPrinters();
TStringList *GetNetworkPrinters();
TStringList *GetLocalMonitors();
TStringList *GetLocalPorts();
TStringList *GetConfigFileList();
TStringList *LoadPrinterInfoFromFile(AnsiString PrinterName);
AnsiString GetDefaultPrinter();

TStringList *Messages;
bool DeletePrinterConfig(AnsiString PrinterConfigName);
AnsiString GetPrinterShareName(AnsiString PrinterName);
AnsiString GetPrinterFullName(AnsiString PrinterName);
bool ClearNetworkPrinters();
bool SetIcaPrinterRights();
bool CopyConfiguration(AnsiString Source, AnsiString Destination);
bool SaveLocalDriver(AnsiString DriverName);

```

```

__published:

```

```

};

```

```

//-----
#endif

```

```

PrinterControl.Cpp

```

```

//-----
#include <vcl.h>
#pragma hdrstop
#pragma warn -aus

```

```

#include "PrinterControl.h"
#pragma package(smart_init)

```

```

typedef bool (*ADDPORTEX)(LPWSTR, DWORD, LPBYTE, LPWSTR);

```

```

//-----
// ValidCtrCheck is used to assure that the components created do not have
// any pure virtual functions.
//

```

```

static inline void ValidCtrCheck(TPrinterControl *)

```

```

{
    new TPrinterControl(NULL, NULL);
}

```

```

//-----

```

```

__fastcall TPrinterControl::TPrinterControl(AnsiString PathToPrinterInfoFiles,

```

FIG. 6.15

```

    AnsiString SourceServerNameForDrivers)
: TComponent(NULL)
{
    SelectedPrinterInfo = new PRINTER_INFO_2;
    ZeroMemory(SelectedPrinterInfo, sizeof(*SelectedPrinterInfo));

    PrtInfoPath = PathToPrinterInfoFiles;

    if (SourceServerNameForDrivers.IsEmpty())
    {
        SourceServerName = "\\\\";
        SourceServerName = SourceServerName + getenv("COMPUTERNAME");
    }
    else if (0 == SourceServerNameForDrivers.SubString(0, 2).AnsiCompareIC("\\\\"))
    {
        SourceServerName = "\\\\" + SourceServerNameForDrivers;
    }
    else
    {
        SourceServerName = SourceServerNameForDrivers;
    }

    Messages = new TStringList;

}

fastcall TPrinterControl::~TPrinterControl()
{
    if (SelectedPrinterInfo)
        free(SelectedPrinterInfo);
    SelectedPrinterInfo = NULL;
    Messages->Free();
}

//-----
namespace Printercontrol
{
    void __fastcall PACKAGE Register()
    {
        TComponentClass classes[1] = {__classid(TPrinterControl)};
        RegisterComponents("Tricerat", classes, 0);
    }
}

//-----

TStringList *TPrinterControl::GetLocalDrivers()
{
    TStringList *LocalDriverList = new TStringList;
    DRIVER_INFO_3 *InstalledDriverInfo = new DRIVER_INFO_3;
    DWORD InstalledDriverInfoReturned;
    DWORD dwSize;
    DWORD dwNeeded;
    int i;

    EnumPrinterDrivers(NULL, NULL, 3, (unsigned char*)InstalledDriverInfo,
        0, &dwSize, &InstalledDriverInfoReturned);

    InstalledDriverInfo = (DRIVER_INFO_3*)malloc(dwSize);

```

FIG. 6.16

```

ZeroMemory(InstalledDriverInfo, dwSize);

if (!EnumPrinterDrivers(NULL, NULL, 3, (unsigned char*)InstalledDriverInfo,
    dwSize, &dwNeeded, &InstalledDriverInfoReturned))
{
    Messages->Add("EnumPrinterDrivers() Failed!");
}

i = -1;
LocalDriverList->Clear();
while ((int)InstalledDriverInfoReturned > ++i)
    LocalDriverList->Add(InstalledDriverInfo[i].pName);

free(InstalledDriverInfo);
return LocalDriverList;
}

TStringList *TPrinterControl::GetLocalPrinters()
{
    TStringList *LocalPrinterList = new TStringList;
    PRINTER_INFO_2 *InstalledPrinterInfo = new PRINTER_INFO_2;
    DWORD InstalledPrinterInfoReturned;
    DWORD dwSize;
    DWORD dwNeeded;
    int i;

    EnumPrinters(PRINTER_ENUM_LOCAL, NULL, 2, (BYTE*)InstalledPrinterInfo,
        0, &dwSize, &InstalledPrinterInfoReturned);

    InstalledPrinterInfo = (PRINTER_INFO_2*)malloc(dwSize);
    ZeroMemory(InstalledPrinterInfo, dwSize);

    if (!EnumPrinters(PRINTER_ENUM_LOCAL, NULL, 2, (BYTE*)InstalledPrinterInfo,
        dwSize, &dwNeeded, &InstalledPrinterInfoReturned))
    {
        Messages->Add("EnumPrinters() Failed!");
    }

    i = -1;
    LocalPrinterList->Clear();
    while ((int)InstalledPrinterInfoReturned > ++i)
        LocalPrinterList->Add(InstalledPrinterInfo[i].pPrinterName);

    free(InstalledPrinterInfo);
    return LocalPrinterList;
}

TStringList *TPrinterControl::GetNetworkPrinters()
{
    TStringList *NetworkPrinterList = new TStringList;
    PRINTER_INFO_2 *InstalledPrinterInfo = new PRINTER_INFO_2;
    DWORD InstalledPrinterInfoReturned;
    DWORD dwSize;
    DWORD dwNeeded;
    int i;

```

FIG. 6.17

```

EnumPrinters(PRINTER_ENUM_CONNECTIONS, NULL, 2, (BYTE*)InstalledPrinterInfo,
0, &dwSize, &InstalledPrinterInfoReturned);

InstalledPrinterInfo = (PRINTER_INFO_2*)malloc(dwSize);
ZeroMemory(InstalledPrinterInfo, dwSize);

if (!EnumPrinters(PRINTER_ENUM_CONNECTIONS, NULL, 2, (BYTE*)InstalledPrinterInfo,
dwSize, &dwNeeded, &InstalledPrinterInfoReturned))
{
    Messages->Add("EnumPrinters() Failed!");
}

i = -1;
NetworkPrinterList->Clear();
while ((int)InstalledPrinterInfoReturned > ++i)
    NetworkPrinterList->Add(InstalledPrinterInfo[i].pPrinterName);

free(InstalledPrinterInfo);
return NetworkPrinterList;
}

AnsiString TPrinterControl::GetDefaultPrinter()
{
    char szPrinter[256];
    AnsiString DefaultPrinter;
    int nDelim;

    GetProfileString("windows", "device", "", szPrinter, sizeof(szPrinter));
    DefaultPrinter = szPrinter;

    nDelim = DefaultPrinter.Pos(",");

    DefaultPrinter = DefaultPrinter.SubString(1, nDelim - 1);

    return DefaultPrinter;
}

TStringList *TPrinterControl::GetLocalMonitors()
{
    MONITOR_INFO_2 *pLocalMonitors = new MONITOR_INFO_2;
    TStringList *LocalMonitors = new TStringList;
    DWORD dwSize;
    DWORD dwBytesNeeded;
    DWORD dwReturned;
    int i;

    //Get the memory needed.
    EnumMonitors(NULL, 2, NULL, 0, &dwSize, &dwReturned);
    pLocalMonitors = (MONITOR_INFO_2*)malloc(dwSize);

    if (!EnumMonitors(NULL, 2, (unsigned char*)pLocalMonitors, dwSize, &dwBytesNeeded,
    &dwReturned))
    {
        Messages->Add("EnumMonitors() Failed!");
    }
}

```

FIG. 6.18

```

i = -1;
while ((int)dwReturned > ++i)
    LocalMonitors->Add(pLocalMonitors[i].pName);

free(pLocalMonitors);

return LocalMonitors;
}

TStringList *TPrinterControl::GetLocalPorts()
{
    PORT_INFO_1 *pLocalPorts = new PORT_INFO_1;
    TStringList *LocalPorts = new TStringList;
    DWORD dwSize;
    DWORD dwReturned;
    DWORD dwBytesNeeded;
    int i;

    EnumPorts(NULL, 1, (unsigned char*)pLocalPorts, 0, &dwSize, &dwReturned);
    pLocalPorts = (PORT_INFO_1*)malloc(dwSize);

    if (!EnumPorts(NULL, 1, (unsigned char*)pLocalPorts, dwSize, &dwBytesNeeded,
        &dwReturned))
    {
        Messages->Add("EnumPorts() Failed!");
    }

    i = -1;
    while ((int)dwReturned > ++i)
        LocalPorts->Add(pLocalPorts[i].pName);

    free(pLocalPorts);

    return LocalPorts;
}

AnsiString TPrinterControl::GetPortMonitor(AnsiString PortName)
{
    PORT_INFO_2 *pPortInfo = new PORT_INFO_2;
    DWORD dwBytesNeeded;
    DWORD dwSize;
    DWORD dwReturned;
    int i;
    AnsiString MonitorName;
    AnsiString LprPortPath;
    TRegistry *Reg = new TRegistry;

    EnumPorts(NULL, 2, (unsigned char*)pPortInfo, 0, &dwSize, &dwReturned);

    pPortInfo = (PORT_INFO_2*)malloc(dwSize);

    if (!EnumPorts(NULL, 2, (unsigned char*)pPortInfo, dwSize, &dwBytesNeeded,
        &dwReturned))
    {
        Messages->Add("EnumPorts() Failed!");
    }
}

```

FIG. 6.19


```

i = -1;
while ((int)dwReturned > ++i)
{
    if (0 == strcmp(PortName.c_str(), pPortInfo[i].pPortName))
        MonitorName = pPortInfo[i].pDescription;
}

free(pPortInfo);

if (MonitorName.IsEmpty())
{
    //Check for LPR Port.
    Reg->RootKey = HKEY_LOCAL_MACHINE;

    LprPortPath = "SYSTEM\\CurrentControlSet\\Control\\Print\\";
    LprPortPath = LprPortPath + "Monitors\\LPR Port\\Ports\\";
    LprPortPath = LprPortPath + PortName;

    if (Reg->OpenKey(LprPortPath, false))
        MonitorName = "LPR Port";
}

Reg->CloseKey();

return MonitorName;
}

bool TPrinterControl::SetDefaultPrinter(AnsiString PrinterToSetAsDefault)
{
    PrinterName = PrinterToSetAsDefault;
    if (!SetDefaultPrinter())
    {
        Messages->Add("SetDefaultPrinter() Failed!");
        return false;
    }

    return true;
}

bool TPrinterControl::SetDefaultPrinter()
{
    HANDLE hPrinter;
    DWORD dwNeeded, dwReturned;
    PRINTER_INFO_2* pPrtInfo;
    char szTemp[256];
    AnsiString szPort;

    //Open handle to printer.
    if (!OpenPrinter(PrinterName.c_str(), &hPrinter, NULL))
    {
        Messages->Add("OpenPrinter() Failed!");
        return false;
    }

    //Select the default printer.

```

FIG. 6.20

```

        if(NULL!=hPrinter){

            // Get the buffer size needed
            GetPrinter(hPrinter,2,NULL,0,&dwNeeded);

            pPrtInfo=(PRINTER_INFO_2*)malloc(dwNeeded);
            ZeroMemory(pPrtInfo, dwNeeded);

            //get the printer info
            GetPrinter(hPrinter,2,(unsigned char*)pPrtInfo,dwNeeded,&dwReturned);

            szPort=pPrtInfo->pPortName;

            //Set the default printer.
            sprintf(szTemp,"%s,WINSPOOL,%s", PrinterName.c_str(), szPort.c_str());
            WriteProfileString("windows","device",szTemp);
            SendNotifyMessage(HWND_BROADCAST, WM_WININICHANGE, 0, 0L);

            //Close the handle to the printer.
            ClosePrinter(hPrinter);
        }

        free(pPrtInfo);

        return true;
    }

bool TPrinterControl::WritePrinterInfo(AnsiString FileToSaveTo)
{
    HANDLE hFile;
    DWORD dwBytesWritten;
    DWORD dwServerNameSize,
        dwPrinterNameSize,
        dwShareNameSize,
        dwPortNameSize,
        dwDriverNameSize,
        dwCommentSize,
        dwLocationSize,
        dwSepFileSize,
        dwPrintProcessorSize,
        dwDatatypeSize,
        dwParametersSize,
        dwPortMonitorSize;

    hFile = CreateFile(FileToSaveTo.c_str(), GENERIC_WRITE, NULL, NULL,
        CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);

    if (NULL == hFile)
    {
        Messages->Add("CreateFile() Failed!");
        return false;
    }

    PortMonitorDescription = GetPortMonitor(SelectedPrinterInfo->pPortName);

    //Set the port to Local if not recognized.

```

FIG. 6.21

```

if (PortMonitorDescription.IsLocalPort())
{
    PortMonitorDescription = "Local Port";
    SelectedPrinterInfo->pPortName = "LPT1:";
}

SetFilePointer(hFile, 0, 0, FILE_BEGIN);

//dwServerNameSize
if (NULL == SelectedPrinterInfo->pServerName)
    dwServerNameSize = 0;
else
    dwServerNameSize = strlen(SelectedPrinterInfo->pServerName);

//dwPrinterNameSize
if (NULL == SelectedPrinterInfo->pPrinterName)
    dwPrinterNameSize = 0;
else
    dwPrinterNameSize = strlen(SelectedPrinterInfo->pPrinterName);

//dwShareNameSize
if (NULL == SelectedPrinterInfo->pShareName)
    dwShareNameSize = 0;
else
    dwShareNameSize = strlen(SelectedPrinterInfo->pShareName);

//dwPortNameSize
if (NULL == SelectedPrinterInfo->pPortName)
    dwPortNameSize = 0;
else
    dwPortNameSize = strlen(SelectedPrinterInfo->pPortName);

//dwDriverNameSize
if (NULL == SelectedPrinterInfo->pDriverName)
    dwDriverNameSize = 0;
else
    dwDriverNameSize = strlen(SelectedPrinterInfo->pDriverName);

//dwCommentSize
if (NULL == SelectedPrinterInfo->pComment)
    dwCommentSize = 0;
else
    dwCommentSize = strlen(SelectedPrinterInfo->pComment);

//dwLocationSize
if (NULL == SelectedPrinterInfo->pLocation)
    dwLocationSize = 0;
else
    dwLocationSize = strlen(SelectedPrinterInfo->pLocation);

//dwSepFileSize
if (NULL == SelectedPrinterInfo->pSepFile)
    dwSepFileSize = 0;
else
    dwSepFileSize = strlen(SelectedPrinterInfo->pSepFile);

//dwPrintProcessorSize

```

FIG. 6.22

```

if (NULL == SelectedPrinterInfo->pPrintProcessor)
    dwPrintProcessorSize = 0;
else
    dwPrintProcessorSize = strlen(SelectedPrinterInfo->pPrintProcessor);

//dwDatatypeSize
if (NULL == SelectedPrinterInfo->pDatatype)
    dwDatatypeSize = 0;
else
    dwDatatypeSize = strlen(SelectedPrinterInfo->pDatatype);

//dwParametersSize
if (NULL == SelectedPrinterInfo->pParameters)
    dwParametersSize = 0;
else
    dwParametersSize = strlen(SelectedPrinterInfo->pParameters);

//dwPortMonitorSize
if (PortMonitorDescription.IsEmpty())
    dwPortMonitorSize = 0;
else
    dwPortMonitorSize = strlen(PortMonitorDescription.c_str());

//Increment the sizes to account for null terminators.
dwServerNameSize++;
dwPrinterNameSize++;
dwShareNameSize++;
dwPortNameSize++;
dwDriverNameSize++;
dwCommentSize++;
dwLocationSize++;
dwSepFileSize++;
dwPrintProcessorSize++;
dwDatatypeSize++;
dwParametersSize++;
dwPortMonitorSize++;

//Write the header.

//dwSelectedPrinterInfoSize
WriteFile(hFile, (char*)&SelectedPrinterInfoSize,
    sizeof(DWORD), &dwBytesWritten, NULL);

//dwServerNameSize
WriteFile(hFile, (char*)&dwServerNameSize,
    sizeof(DWORD), &dwBytesWritten, NULL);

//dwPrinterNameSize
WriteFile(hFile, (char*)&dwPrinterNameSize,
    sizeof(DWORD), &dwBytesWritten, NULL);

//dwShareNameSize
WriteFile(hFile, (char*)&dwShareNameSize,
    sizeof(DWORD), &dwBytesWritten, NULL);

//dwPortNameSize

```

FIG. 6.23

```

WriteFile(hFile, (char*)&dwPortNameSize,
    sizeof(DWORD), &dwBytesWritten, NULL);

//dwDriverNameSize
WriteFile(hFile, (char*)&dwDriverNameSize,
    sizeof(DWORD), &dwBytesWritten, NULL);

//dwCommentSize
WriteFile(hFile, (char*)&dwCommentSize,
    sizeof(DWORD), &dwBytesWritten, NULL);

//dwLocationSize
WriteFile(hFile, (char*)&dwLocationSize,
    sizeof(DWORD), &dwBytesWritten, NULL);

//dwSepFileSize
WriteFile(hFile, (char*)&dwSepFileSize,
    sizeof(DWORD), &dwBytesWritten, NULL);

//dwPrintProcessorSize
WriteFile(hFile, (char*)&dwPrintProcessorSize,
    sizeof(DWORD), &dwBytesWritten, NULL);

//dwDatatypeSize
WriteFile(hFile, (char*)&dwDatatypeSize,
    sizeof(DWORD), &dwBytesWritten, NULL);

//dwParametersSize
WriteFile(hFile, (char*)&dwParametersSize,
    sizeof(DWORD), &dwBytesWritten, NULL);

//dwPortMonitorSize
WriteFile(hFile, (char*)&dwPortMonitorSize,
    sizeof(DWORD), &dwBytesWritten, NULL);

//Write the data.

//pServerName
if (NULL == SelectedPrinterInfo->pServerName)
    WriteFile(hFile, (char*)"",
        dwServerNameSize, &dwBytesWritten, NULL);
else
    WriteFile(hFile, (char*)SelectedPrinterInfo->pServerName,
        dwServerNameSize, &dwBytesWritten, NULL);

//pPrinterName
if (NULL == SelectedPrinterInfo->pPrinterName)
    WriteFile(hFile, (char*)"",
        dwPrinterNameSize, &dwBytesWritten, NULL);
else
    WriteFile(hFile, (char*)SelectedPrinterInfo->pPrinterName,
        dwPrinterNameSize, &dwBytesWritten, NULL);

//pShareName
if (NULL == SelectedPrinterInfo->pShareName)
    WriteFile(hFile, (char*)"",
        dwShareNameSize, &dwBytesWritten, NULL);

```

FIG. 6.24

```

else
    WriteFile(hFile, (char*)SelectedPrinterInfo->pShareName,
        dwShareNameSize, &dwBytesWritten, NULL);

//pPortName
if (NULL == SelectedPrinterInfo->pPortName)
    WriteFile(hFile, (char*)"",
        dwPortNameSize, &dwBytesWritten, NULL);
else
    WriteFile(hFile, (char*)SelectedPrinterInfo->pPortName,
        dwPortNameSize, &dwBytesWritten, NULL);

//pDriverName
if (NULL == SelectedPrinterInfo->pDriverName)
    WriteFile(hFile, (char*)"",
        dwDriverNameSize, &dwBytesWritten, NULL);
else
    WriteFile(hFile, (char*)SelectedPrinterInfo->pDriverName,
        dwDriverNameSize, &dwBytesWritten, NULL);

//pComment
if (NULL == SelectedPrinterInfo->pComment)
    WriteFile(hFile, (char*)"",
        dwCommentSize, &dwBytesWritten, NULL);
else
    WriteFile(hFile, (char*)SelectedPrinterInfo->pComment,
        dwCommentSize, &dwBytesWritten, NULL);

//pLocation
if (NULL == SelectedPrinterInfo->pLocation)
    WriteFile(hFile, (char*)"",
        dwLocationSize, &dwBytesWritten, NULL);
else
    WriteFile(hFile, (char*)SelectedPrinterInfo->pLocation,
        dwLocationSize, &dwBytesWritten, NULL);

//pSepFile
if (NULL == SelectedPrinterInfo->pSepFile)
    WriteFile(hFile, (char*)"",
        dwSepFileSize, &dwBytesWritten, NULL);
else
    WriteFile(hFile, (char*)SelectedPrinterInfo->pSepFile,
        dwSepFileSize, &dwBytesWritten, NULL);

//pPrintProcessor
if (NULL == SelectedPrinterInfo->pPrintProcessor)
    WriteFile(hFile, (char*)"",
        dwPrintProcessorSize, &dwBytesWritten, NULL);
else
    WriteFile(hFile, (char*)SelectedPrinterInfo->pPrintProcessor,
        dwPrintProcessorSize, &dwBytesWritten, NULL);

//pDatatype
if (NULL == SelectedPrinterInfo->pDatatype)
    WriteFile(hFile, (char*)"",
        dwDatatypeSize, &dwBytesWritten, NULL);
else

```

FIG. 6.25

```

WriteFile(hFile, (char*)SelectedPrinterInfo->pDatatype,
    dwDatatypeSize, &dwBytesWritten, NULL);

//pParameters
if (NULL == SelectedPrinterInfo->pParameters)
    WriteFile(hFile, (char*)"",
        dwParametersSize, &dwBytesWritten, NULL);
else
    WriteFile(hFile, (char*)SelectedPrinterInfo->pParameters,
        dwParametersSize, &dwBytesWritten, NULL);

//pPortMonitorName
if (PortMonitorDescription.IsEmpty())
    WriteFile(hFile, (char*)"",
        dwPortMonitorSize, &dwBytesWritten, NULL);
else
    WriteFile(hFile, (char*)PortMonitorDescription.c_str(),
        dwPortMonitorSize, &dwBytesWritten, NULL);

//Attributes
WriteFile(hFile, (CHAR*)&SelectedPrinterInfo->Attributes,
    sizeof(DWORD), &dwBytesWritten, NULL);

//Priority
WriteFile(hFile, (char*)&SelectedPrinterInfo->Priority,
    sizeof(DWORD), &dwBytesWritten, NULL);

//DefaultPriority
WriteFile(hFile, (char*)&SelectedPrinterInfo->DefaultPriority,
    sizeof(DWORD), &dwBytesWritten, NULL);

//StartTime
WriteFile(hFile, (char*)&SelectedPrinterInfo->StartTime,
    sizeof(DWORD), &dwBytesWritten, NULL);

//UntilTime
WriteFile(hFile, (char*)&SelectedPrinterInfo->UntilTime,
    sizeof(DWORD), &dwBytesWritten, NULL);

//Status
WriteFile(hFile, (char*)&SelectedPrinterInfo->Status,
    sizeof(DWORD), &dwBytesWritten, NULL);

//cJobs
WriteFile(hFile, (char*)&SelectedPrinterInfo->cJobs,
    sizeof(DWORD), &dwBytesWritten, NULL);

//AveragePPM
WriteFile(hFile, (char*)&SelectedPrinterInfo->AveragePPM,
    sizeof(DWORD), &dwBytesWritten, NULL);

//Now write the DevMode structure.

//Entire structure size.
WriteFile(hFile, (char*)&dwDevModeSize,
    sizeof(DWORD), &dwBytesWritten, NULL);

```

FIG. 6.26

```

//dmSize
WriteFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmSize,
    sizeof(WORD), &dwBytesWritten, NULL);

//dmDeviceName[32]
WriteFile(hFile, (char*)SelectedPrinterInfo->pDevMode->dmDeviceName,
    CCHDEVICENAME, &dwBytesWritten, NULL);

//dmSpecVersion
WriteFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmSpecVersion,
    sizeof(WORD), &dwBytesWritten, NULL);

//dmDriverVersion
WriteFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmDriverVersion,
    sizeof(WORD), &dwBytesWritten, NULL);

//dmDriverExtra
WriteFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmDriverExtra,
    sizeof(WORD), &dwBytesWritten, NULL);

//dmFields
WriteFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmFields,
    sizeof(DWORD), &dwBytesWritten, NULL);

//dmOrientation
WriteFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmOrientation,
    sizeof(short), &dwBytesWritten, NULL);

//dmPaperSize
WriteFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmPaperSize,
    sizeof(short), &dwBytesWritten, NULL);

//dmPaperLength
WriteFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmPaperLength,
    sizeof(short), &dwBytesWritten, NULL);

//dmPaperWidth
WriteFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmPaperWidth,
    sizeof(short), &dwBytesWritten, NULL);

//dmScale
WriteFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmScale,
    sizeof(short), &dwBytesWritten, NULL);

//dmCopies
WriteFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmCopies,
    sizeof(short), &dwBytesWritten, NULL);

//dmDefaultSource
WriteFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmDefaultSource,
    sizeof(short), &dwBytesWritten, NULL);

//dmPrintQuality
WriteFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmPrintQuality,
    sizeof(short), &dwBytesWritten, NULL);

//dmColor

```

FIG. 6.27


```

WriteFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmColor,
    sizeof(short), &dwBytesWritten, NULL);

//dmDuplex
WriteFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmDuplex,
    sizeof(short), &dwBytesWritten, NULL);

//dmYResolution
WriteFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmYResolution,
    sizeof(short), &dwBytesWritten, NULL);

//dmTTOption
WriteFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmTTOption,
    sizeof(short), &dwBytesWritten, NULL);

//dmCollate
WriteFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmCollate,
    sizeof(short), &dwBytesWritten, NULL);

//dmFormName[32]
WriteFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmFormName,
    CCHFORMNAME, &dwBytesWritten, NULL);

//dmBitsPerPel
WriteFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmBitsPerPel,
    sizeof(USHORT), &dwBytesWritten, NULL);

//dmPelsWidth
WriteFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmPelsWidth,
    sizeof(DWORD), &dwBytesWritten, NULL);

//dmPelsHeight
WriteFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmPelsHeight,
    sizeof(DWORD), &dwBytesWritten, NULL);

//dmDisplayFlags
WriteFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmDisplayFlags,
    sizeof(DWORD), &dwBytesWritten, NULL);

//dmDisplayFrequency
WriteFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmDisplayFrequency,
    sizeof(DWORD), &dwBytesWritten, NULL);

CloseHandle(hFile);

return true;
}

bool TPrinterControl::ReadPrinterInfo(AnsiString FileToReadFrom)
{
    HANDLE hFile;
    DWORD dwBytesRead;
    DWORD dwServerNameSize,
        dwPrinterNameSize,
        dwShareNameSize,
        dwPortNameSize,
        dwDriverNameSize,

```

FIG. 6.28

```

    dwCommentSize,
    dwLocationSize,
    dwSepFileSize,
    dwPrintProcessorSize,
    dwDatatypeSize,
    dwParametersSize,
    dwPortMonitorSize;
void *pPortMonitorName;

hFile = CreateFile(FileToReadFrom.c_str(), GENERIC_READ,
    FILE_SHARE_READ, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

if (NULL == hFile)
{
    Messages->Add("CreateFile() Failed!");
    return false;
}

SetFilePointer(hFile, 0, 0, FILE_BEGIN);

//PrinterInfoSize
SelectedPrinterInfoSize = 0;
ReadFile(hFile, (char*)&SelectedPrinterInfoSize,
    sizeof(DWORD), &dwBytesRead, NULL);

//dwServerNameSize
dwServerNameSize = 0;
ReadFile(hFile, (char*)&dwServerNameSize,
    sizeof(DWORD), &dwBytesRead, NULL);

//dwPrinterNameSize
dwPrinterNameSize = 0;
ReadFile(hFile, (char*)&dwPrinterNameSize,
    sizeof(DWORD), &dwBytesRead, NULL);

//dwShareNameSize
dwShareNameSize = 0;
ReadFile(hFile, (char*)&dwShareNameSize,
    sizeof(DWORD), &dwBytesRead, NULL);

//dwPortNameSize
dwPortNameSize = 0;
ReadFile(hFile, (char*)&dwPortNameSize,
    sizeof(DWORD), &dwBytesRead, NULL);

//dwDriverNameSize
dwDriverNameSize = 0;
ReadFile(hFile, (char*)&dwDriverNameSize,
    sizeof(DWORD), &dwBytesRead, NULL);

//dwCommentSize
dwCommentSize = 0;
ReadFile(hFile, (char*)&dwCommentSize,
    sizeof(DWORD), &dwBytesRead, NULL);

//dwLocationSize
dwLocationSize = 0;

```

FIG. 6.29

```
ReadFile(hFile, (char*)&dwLongNameSize,  
sizeof(DWORD), &dwBytesRead, NULL);
```

```
//dwSepFileSize  
dwSepFileSize = 0;  
ReadFile(hFile, (char*)&dwSepFileSize,  
sizeof(DWORD), &dwBytesRead, NULL);
```

```
//dwPrintProcessorSize  
dwPrintProcessorSize = 0;  
ReadFile(hFile, (char*)&dwPrintProcessorSize,  
sizeof(DWORD), &dwBytesRead, NULL);
```

```
//dwDatatypeSize  
dwDatatypeSize = 0;  
ReadFile(hFile, (char*)&dwDatatypeSize,  
sizeof(DWORD), &dwBytesRead, NULL);
```

```
//dwParametersSize  
dwParametersSize = 0;  
ReadFile(hFile, (char*)&dwParametersSize,  
sizeof(DWORD), &dwBytesRead, NULL);
```

```
//dwPortMonitorSize  
dwPortMonitorSize = 0;  
ReadFile(hFile, (char*)&dwPortMonitorSize,  
sizeof(DWORD), &dwBytesRead, NULL);
```

```
free(SelectedPrinterInfo);  
SelectedPrinterInfo = NULL;  
SelectedPrinterInfo = (PRINTER_INFO_2*)malloc(SelectedPrinterInfoSize);  
ZeroMemory(SelectedPrinterInfo, SelectedPrinterInfoSize);
```

```
SelectedPrinterInfo->pServerName = NULL;  
SelectedPrinterInfo->pServerName = (LPTSTR)malloc(dwServerNameSize);  
ZeroMemory(SelectedPrinterInfo->pServerName, dwServerNameSize);
```

```
SelectedPrinterInfo->pPrinterName = NULL;  
SelectedPrinterInfo->pPrinterName = (LPTSTR)malloc(dwPrinterNameSize);  
ZeroMemory(SelectedPrinterInfo->pPrinterName, dwPrinterNameSize);
```

```
SelectedPrinterInfo->pShareName = NULL;  
SelectedPrinterInfo->pShareName = (LPTSTR)malloc(dwShareNameSize);  
ZeroMemory(SelectedPrinterInfo->pShareName, dwShareNameSize);
```

```
SelectedPrinterInfo->pPortName = NULL;  
SelectedPrinterInfo->pPortName = (LPTSTR)malloc(dwPortNameSize);  
ZeroMemory(SelectedPrinterInfo->pPortName, dwPortNameSize);
```

```
SelectedPrinterInfo->pDriverName = NULL;  
SelectedPrinterInfo->pDriverName = (LPTSTR)malloc(dwDriverNameSize);  
ZeroMemory(SelectedPrinterInfo->pDriverName, dwDriverNameSize);
```

```
SelectedPrinterInfo->pComment = NULL;  
SelectedPrinterInfo->pComment = (LPTSTR)malloc(dwCommentSize);  
ZeroMemory(SelectedPrinterInfo->pComment, dwCommentSize);
```

FIG. 6.30

```

SelectedPrinterInfo->pLocation = NULL;
SelectedPrinterInfo->pLocation = (LPTSTR)malloc(dwLocationSize);
ZeroMemory(SelectedPrinterInfo->pLocation, dwLocationSize);

SelectedPrinterInfo->pSepFile = NULL;
SelectedPrinterInfo->pSepFile = (LPTSTR)malloc(dwSepFileSize);
ZeroMemory(SelectedPrinterInfo->pSepFile, dwSepFileSize);

SelectedPrinterInfo->pPrintProcessor = NULL;
SelectedPrinterInfo->pPrintProcessor = (LPTSTR)malloc(dwPrintProcessorSize);
ZeroMemory(SelectedPrinterInfo->pPrintProcessor, dwPrintProcessorSize);

SelectedPrinterInfo->pDatatype = NULL;
SelectedPrinterInfo->pDatatype = (LPTSTR)malloc(dwDatatypeSize);
ZeroMemory(SelectedPrinterInfo->pDatatype, dwDatatypeSize);

SelectedPrinterInfo->pParameters = NULL;
SelectedPrinterInfo->pParameters = (LPTSTR)malloc(dwParametersSize);
ZeroMemory(SelectedPrinterInfo->pParameters, dwParametersSize);

pPortMonitorName = NULL;
pPortMonitorName = malloc(dwPortMonitorSize);
ZeroMemory(pPortMonitorName, dwPortMonitorSize);

SelectedPrinterInfo->Attributes = (DWORD)malloc(sizeof(DWORD));
SelectedPrinterInfo->Attributes = 0;

SelectedPrinterInfo->Priority = (DWORD)malloc(sizeof(DWORD));
SelectedPrinterInfo->Priority = 0;

SelectedPrinterInfo->DefaultPriority = (DWORD)malloc(sizeof(DWORD));
SelectedPrinterInfo->DefaultPriority = 0;

SelectedPrinterInfo->StartTime = (DWORD)malloc(sizeof(DWORD));
SelectedPrinterInfo->StartTime = 0;

SelectedPrinterInfo->UntilTime = (DWORD)malloc(sizeof(DWORD));
SelectedPrinterInfo->UntilTime = 0;

SelectedPrinterInfo->Status = (DWORD)malloc(sizeof(DWORD));
SelectedPrinterInfo->Status = 0;

SelectedPrinterInfo->cJobs = (DWORD)malloc(sizeof(DWORD));
SelectedPrinterInfo->cJobs = 0;

SelectedPrinterInfo->AveragePPM = (DWORD)malloc(sizeof(DWORD));
SelectedPrinterInfo->AveragePPM = 0;

//pServerName
ReadFile(hFile, (char*)SelectedPrinterInfo->pServerName,
        dwServerNameSize, &dwBytesRead, NULL);

//pPrinterName
ReadFile(hFile, (char*)SelectedPrinterInfo->pPrinterName,
        dwPrinterNameSize, &dwBytesRead, NULL);

//pShareName

```

FIG. 6.31

```

ReadFile(hFile, (char*)SelectedPrinterInfo->pShareName,
        dwShareNameSize, &dwBytesRead, NULL);

//pPortName
ReadFile(hFile, (char*)SelectedPrinterInfo->pPortName,
        dwPortNameSize, &dwBytesRead, NULL);

//pDriverName
ReadFile(hFile, (char*)SelectedPrinterInfo->pDriverName,
        dwDriverNameSize, &dwBytesRead, NULL);

//pComment
ReadFile(hFile, (char*)SelectedPrinterInfo->pComment,
        dwCommentSize, &dwBytesRead, NULL);

//pLocation
ReadFile(hFile, (char*)SelectedPrinterInfo->pLocation,
        dwLocationSize, &dwBytesRead, NULL);

//pSepFile
ReadFile(hFile, (char*)SelectedPrinterInfo->pSepFile,
        dwSepFileSize, &dwBytesRead, NULL);

//pPrintProcessor
ReadFile(hFile, (char*)SelectedPrinterInfo->pPrintProcessor,
        dwPrintProcessorSize, &dwBytesRead, NULL);

//pDatatype
ReadFile(hFile, (char*)SelectedPrinterInfo->pDatatype,
        dwDatatypeSize, &dwBytesRead, NULL);

//pParameters
ReadFile(hFile, (char*)SelectedPrinterInfo->pParameters,
        dwParametersSize, &dwBytesRead, NULL);

//pPortMonitorName
ReadFile(hFile, (char*)pPortMonitorName,
        dwPortMonitorSize, &dwBytesRead, NULL);
PortMonitorDescription = (char*)pPortMonitorName;

//Attributes
ReadFile(hFile, (char*)&SelectedPrinterInfo->Attributes,
        sizeof(DWORD), &dwBytesRead, NULL);

//Priority
ReadFile(hFile, (char*)&SelectedPrinterInfo->Priority,
        sizeof(DWORD), &dwBytesRead, NULL);

//DefaultPriority
ReadFile(hFile, (char*)&SelectedPrinterInfo->DefaultPriority,
        sizeof(DWORD), &dwBytesRead, NULL);

//StartTime
ReadFile(hFile, (char*)&SelectedPrinterInfo->StartTime,
        sizeof(DWORD), &dwBytesRead, NULL);

//UntilTime

```

FIG. 6.32

```

ReadFile(hFile, (char*)&SelectedPrinterInfo->UntilTime,
    sizeof(DWORD), &dwBytesRead, NULL);

//Status
ReadFile(hFile, (char*)&SelectedPrinterInfo->Status,
    sizeof(DWORD), &dwBytesRead, NULL);

//cJobs
ReadFile(hFile, (char*)&SelectedPrinterInfo->cJobs,
    sizeof(DWORD), &dwBytesRead, NULL);

//AveragePPM
ReadFile(hFile, (char*)&SelectedPrinterInfo->AveragePPM,
    sizeof(DWORD), &dwBytesRead, NULL);

//Now read the DevMode Structure size.
ReadFile(hFile, (char*)&dwDevModeSize,
    sizeof(DWORD), &dwBytesRead, NULL);

//Allocate the DevMode structure members.
free(SelectedPrinterInfo->pDevMode);
SelectedPrinterInfo->pDevMode = NULL;
SelectedPrinterInfo->pDevMode = (DEVMODE*)malloc(dwDevModeSize);
ZeroMemory(SelectedPrinterInfo->pDevMode, dwDevModeSize);

ZeroMemory(SelectedPrinterInfo->pDevMode->dmDeviceName, CCHDEVICENAME);
ZeroMemory(SelectedPrinterInfo->pDevMode->dmFormName, CCHFORMNAME);

//dmSize
SelectedPrinterInfo->pDevMode->dmSize = 0;
ReadFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmSize,
    sizeof(WORD), &dwBytesRead, NULL);

//dmDeviceName[32]
ReadFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmDeviceName,
    CCHDEVICENAME, &dwBytesRead, NULL);

//dmSpecVersion
SelectedPrinterInfo->pDevMode->dmSpecVersion = 0;
ReadFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmSpecVersion,
    sizeof(WORD), &dwBytesRead, NULL);

//dmDriverVersion
SelectedPrinterInfo->pDevMode->dmDriverVersion = 0;
ReadFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmDriverVersion,
    sizeof(WORD), &dwBytesRead, NULL);

//dmDriverExtra
SelectedPrinterInfo->pDevMode->dmDriverExtra = 0;
ReadFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmDriverExtra,
    sizeof(WORD), &dwBytesRead, NULL);

//dmFields
SelectedPrinterInfo->pDevMode->dmFields = 0;
ReadFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmFields,
    sizeof(DWORD), &dwBytesRead, NULL);

```

FIG. 6.33

```

//dmOrientation
SelectedPrinterInfo->pDevMode->dmOrientation = 0;
ReadFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmOrientation,
    sizeof(short), &dwBytesRead, NULL);

//dmPaperSize
SelectedPrinterInfo->pDevMode->dmPaperSize = 0;
ReadFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmPaperSize,
    sizeof(short), &dwBytesRead, NULL);

//dmPaperLength
SelectedPrinterInfo->pDevMode->dmPaperLength = 0;
ReadFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmPaperLength,
    sizeof(short), &dwBytesRead, NULL);

//dmPaperWidth
SelectedPrinterInfo->pDevMode->dmPaperWidth = 0;
ReadFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmPaperWidth,
    sizeof(short), &dwBytesRead, NULL);

//dmScale
SelectedPrinterInfo->pDevMode->dmScale = 0;
ReadFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmScale,
    sizeof(short), &dwBytesRead, NULL);

//dmCopies
SelectedPrinterInfo->pDevMode->dmCopies = 0;
ReadFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmCopies,
    sizeof(short), &dwBytesRead, NULL);

//dmDefaultSource
SelectedPrinterInfo->pDevMode->dmDefaultSource = 0;
ReadFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmDefaultSource,
    sizeof(short), &dwBytesRead, NULL);

//dmPrintQuality
SelectedPrinterInfo->pDevMode->dmPrintQuality = 0;
ReadFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmPrintQuality,
    sizeof(short), &dwBytesRead, NULL);

//dmColor
SelectedPrinterInfo->pDevMode->dmColor = 0;
ReadFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmColor,
    sizeof(short), &dwBytesRead, NULL);

//dmDuplex
SelectedPrinterInfo->pDevMode->dmDuplex = 0;
ReadFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmDuplex,
    sizeof(short), &dwBytesRead, NULL);

//dmYResolution
SelectedPrinterInfo->pDevMode->dmYResolution = 0;
ReadFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmYResolution,
    sizeof(short), &dwBytesRead, NULL);

//dmTTOption
SelectedPrinterInfo->pDevMode->dmTTOption = 0;

```

FIG. 6.34

```

ReadFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmTTOption,
    sizeof(short), &dwBytesRead, NULL);

//dmCollate
SelectedPrinterInfo->pDevMode->dmCollate = 0;
ReadFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmCollate,
    sizeof(short), &dwBytesRead, NULL);

//dmFormName[32]
ReadFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmFormName,
    CCHFORMNAME, &dwBytesRead, NULL);

//dmBitsPerPel
SelectedPrinterInfo->pDevMode->dmBitsPerPel = 0;
ReadFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmBitsPerPel,
    sizeof(USHORT), &dwBytesRead, NULL);

//dmPelsWidth
SelectedPrinterInfo->pDevMode->dmPelsWidth = 0;
ReadFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmPelsWidth,
    sizeof(DWORD), &dwBytesRead, NULL);

//dmPelsHeight
SelectedPrinterInfo->pDevMode->dmPelsHeight = 0;
ReadFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmPelsHeight,
    sizeof(DWORD), &dwBytesRead, NULL);

//dmDisplayFlags
SelectedPrinterInfo->pDevMode->dmDisplayFlags = 0;
ReadFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmDisplayFlags,
    sizeof(DWORD), &dwBytesRead, NULL);

//dmDisplayFrequency
SelectedPrinterInfo->pDevMode->dmDisplayFrequency = 0;
ReadFile(hFile, (char*)&SelectedPrinterInfo->pDevMode->dmDisplayFrequency,
    sizeof(DWORD), &dwBytesRead, NULL);

CloseHandle(hFile);
free(pPortMonitorName);
pPortMonitorName = NULL;

return true;
}

bool TPrinterControl::SaveLocalPrinter(AnsiString PrinterToSave, AnsiString SaveName)
{
    PrinterName = PrinterToSave;
    NewPrinterName = SaveName;
    if (!SaveLocalPrinter())
    {
        Messages->Add("SaveLocalPrinter() Failed!");
        return false;
    }

    return true;
}

```

FIG. 6.35


```
bool TPrinterControl::SaveLocalPrinter(AnsiString PrinterToSave)
```

```
{
    PrinterName = PrinterToSave;
    NewPrinterName = PrinterToSave;
    if (!SaveLocalPrinter())
    {
        Messages->Add("SaveLocalPrinter() Failed!");
        return false;
    }

    return true;
}
```

```
bool TPrinterControl::SaveLocalPrinter()
```

```
{
    HANDLE hPrinter;
    DWORD dwReturned;
    AnsiString MonitorName;

    NewPrinterName = CleanupFilename(NewPrinterName);

    //Open handle to printer.
    if( 0 == OpenPrinter(PrinterName.c_str(),&hPrinter,NULL))
    {
        Messages->Add("OpenPrinter() Failed!");
        return false;
    }

    //Select the default printer.
    if(NULL == hPrinter)
    {
        Messages->Add("NULL Printer Handle!");
    }

    // Get the buffer size needed
    GetPrinter(hPrinter,2,NULL,0,&SelectedPrinterInfoSize);

    free(SelectedPrinterInfo);
    SelectedPrinterInfo = (PRINTER_INFO_2*)malloc(SelectedPrinterInfoSize);
    ZeroMemory(SelectedPrinterInfo, SelectedPrinterInfoSize);

    //get the printer info
    if (!GetPrinter(hPrinter, 2, (unsigned char*)SelectedPrinterInfo,
        SelectedPrinterInfoSize, &dwReturned))
    {
        Messages->Add("GetPrinter() Failed!");
    }

    //Get the DevMode structure
    dwDevModeSize = DocumentProperties(NULL, hPrinter,
        PrinterName.c_str(), NULL, NULL, 0);

    SelectedPrinterInfo->pDevMode = (DEVMODE*)malloc(dwDevModeSize);

    DocumentProperties(NULL, hPrinter, PrinterName.c_str(),
        SelectedPrinterInfo->pDevMode, NULL, DM_OUT_BUFFER);
}
```

FIG. 6.36

```

        //Close the handle to the printer.
        ClosePrinter(hPrinter);

        SelectedPrinterInfo->pPrinterName = NewPrinterName.c_str();

        WritePrinterInfo(PrtInfoPath + "\\\" + NewPrinterName + ".Prt");

        TRegTools *RegDump = new TRegTools(HKEY_LOCAL_MACHINE,
            "SYSTEM\\CurrentControlSet\\Control\\Print\\Printers\\" + PrinterName +
            "\\PrinterDriverData", PrtInfoPath + "\\\" + NewPrinterName + ".Dev");
        delete RegDump;
        RegDump = NULL;

        return true;
    }

bool TPrinterControl::CreateLocalPrinter(AnsiString PrinterToCreate,
    AnsiString NewPrinterToCreate)
{
    PrinterName = PrinterToCreate;
    NewPrinterName = NewPrinterToCreate;
    if (!CreateLocalPrinter())
    {
        Messages->Add("CreateLocalPrinter() Failed!");
        return false;
    }

    return true;
}

bool TPrinterControl::CreateLocalPrinter(AnsiString PrinterToCreate)
{
    PrinterName = PrinterToCreate;
    NewPrinterName = PrinterToCreate;
    if (!CreateLocalPrinter())
    {
        Messages->Add("CreateLocalPrinter() Failed!");
        return false;
    }

    return true;
}

bool TPrinterControl::CreateLocalPrinter(AnsiString PrinterToCreate,
    AnsiString NewPrinterToCreate, TStringList *Users)
{
    if (!Users)
        return false;

    PrinterName = PrinterToCreate;
    NewPrinterName = NewPrinterToCreate;

    if (!CreateLocalPrinter())
    {
        Messages->Add("CreateLocalPrinter() Failed!");
        return false;
    }

```

FIG. 6.37

```

    }

    PrinterAddAccessRights(NewPrinterName, Users, CONTROL_FULL);

    return true;
}

bool TPrinterControl::CreateLocalPrinter()
{
    HANDLE hPrinter;
    TStringList *LocalPrinters = new TStringList;
    int i;

STEP 400
    //Read in the PRINTER_INFO_2 structure from file.
    if (!ReadPrinterInfo(PrtInfoPath + "\\\" + PrinterName + ".Prt"))
    {
        Messages->Add("Unable to Read Printer File: " + PrinterName);
        return false;
    }

STEP 410
    SelectedPrinterInfo->pPrinterName = (LPTSTR)malloc(strlen(NewPrinterName.c_str()) + 1);
    SelectedPrinterInfo->pPrinterName = NewPrinterName.c_str();

    LocalPrinters = GetLocalPrinters();

    i = -1;
    while (LocalPrinters->Count > ++i)
    {
        if (0 == strcmp(LocalPrinters->Strings[i].c_str(), NewPrinterName.c_str()))
        {
            LocalPrinters->Free();
            return true;
        }
    }
    LocalPrinters->Free();

    if (!NewPortMonitor.IsEmpty() && !NewPortName.IsEmpty())
    {
        PortMonitorDescription = NewPortMonitor;

        if (0 == NewPortMonitor.AnsiCompareIC("Client Printer Port"))
            NewPortName = GetIcaClientPort(NewPortName);

        SelectedPrinterInfo->pPortName = (LPTSTR)malloc(strlen(NewPortName.c_str()) + 1);
        SelectedPrinterInfo->pPortName = NewPortName.c_str();
    }

STEP 420
    if (!ValidateMonitor(PortMonitorDescription))
    {
        Messages->Add("Invalid Port Monitor: " + PortMonitorDescription);
        return false;
    }

STEP 430

```

FIG. 6.38

```

if (!ValidatePort(SelectedPrinterInfo->pPortName, PortMonitorDescription))
{
    TRegistry *reg = new TRegistry();
    reg->RootKey = HKEY_LOCAL_MACHINE;
    if (reg->OpenKey("Software\\Microsoft\\Windows NT\\CurrentVersion\\Ports", false))
    {
        try
        {
            reg->WriteString("CLIENT\\LPT1:", "");
            reg->WriteString("CLIENT\\LPT2:", "");
            reg->WriteString("CLIENT\\COM1:", "");
            reg->WriteString("CLIENT\\COM2:", "");
        }
        catch(...)
        {
        }
    }
    reg->CloseKey();
    reg->Free();

    if (!ValidatePort(SelectedPrinterInfo->pPortName, PortMonitorDescription))
    {
        Messages->Add("Invalid Port:");
        return false;
    }
}

```

STEP 440

```

if (!ValidateDriver(SelectedPrinterInfo->pDriverName))
{
    Messages->Add("Invalid Driver:");
    return false;
}

```

STEP 450

```

//Add the printer
hPrinter = AddPrinter(NULL, 2, (unsigned char*)SelectedPrinterInfo);

if (NULL == hPrinter)
{
    DWORD dwError = 0;
    dwError = GetLastError();
    Messages->Add("Failed to Install Printer: " + NewPrinterName +
        " Error Number " + String(dwError));
    return false;
}

```

STEP 460

```

DocumentProperties(NULL, hPrinter, NewPrinterName.c_str(),
    SelectedPrinterInfo->pDevMode, SelectedPrinterInfo->pDevMode,
    DM_IN_BUFFER | DM_OUT_BUFFER);

SetPrinter(hPrinter, 2, (BYTE*)SelectedPrinterInfo, 0);

ClosePrinter(hPrinter);

```

FIG. 6.39

```

//Write the Device specific DevMode data. Some drivers do not store this
//in the registry.
TRegistry *Reg = new TRegistry;

Reg->RootKey = HKEY_LOCAL_MACHINE;
if (Reg->OpenKey("SYSTEM\\CurrentControlSet\\Control\\Print\\Printers\\" +
    NewPrinterName, false))
{
    TRegTools *RegDump = new TRegTools(PrtInfoPath + "\\" + PrinterName + ".Dev",
        HKEY_LOCAL_MACHINE,
        "SYSTEM\\CurrentControlSet\\Control\\Print\\Printers\\" +
        NewPrinterName + "\\PrinterDriverData");
    delete RegDump;
    RegDump = NULL;
}

Reg->CloseKey();
Reg->Free();

```

STEP 470

```

PrinterSetCurrentUserOnlyRights(NewPrinterName);
PrinterAddAccessRights(NewPrinterName, "SYSTEM", CONTROL_FULL);

SendNotifyMessage(HWND_BROADCAST, WM_DEVMODECHANGE, 0L,
    (LPARAM)(LPCSTR)NewPrinterName.c_str());

NewPortName = "";
NewPortMonitor = "";

return true;
}

DRIVER_INFO_3 *TPrinterControl::GetRemoteDriverInfo(AnsiString ServerName, AnsiString DriverName)
{
    DWORD dwSize;
    DWORD dwNeeded;
    DWORD dwReturned;
    DRIVER_INFO_3 *pDriverInfoReturn;
    DRIVER_INFO_3 *pDrv = new DRIVER_INFO_3;

    EnumPrinterDrivers(ServerName.c_str(), NULL, 3, (unsigned char*)pDrv,
        0, &dwSize, &dwReturned);

    pDrv = (DRIVER_INFO_3*)malloc(dwSize);
    ZeroMemory(pDrv, dwSize);

    if (!EnumPrinterDrivers(ServerName.c_str(), NULL, 3, (unsigned char*)pDrv,
        dwSize, &dwNeeded, &dwReturned))
    {
        Messages->Add("EnumPrinterDrivers() Failed!");
    }

    int i = -1;
    while ((int)dwReturned > ++i)
    {

```

FIG. 6.40

```

    if (0 == strcmp((const char*)DriverName.c_str(),
        (const char*)pDrv[i].pName))
    {
        pDriverInfoReturn = &pDrv[i];
        break;
    }
}

if ((int)dwReturned <= i)
    return NULL;

return pDriverInfoReturn;
}

```

```

TStringList *TPrinterControl::CopyDriverFiles(TStringList *SourceFiles)
{
    AnsiString LocalDriverDir;
    AnsiString DestFileName;
    TStringList *ReturnStrings = new TStringList;
    BYTE *pTemp;
    DWORD dwBufferSize;
    DWORD dwBytesNeeded;
    int i;

    dwBufferSize = 1024;

    pTemp = (BYTE*)malloc(dwBufferSize);

    if (0 == GetPrinterDriverDirectory(NULL, NULL, 1, pTemp, dwBufferSize,
        &dwBytesNeeded))
        return ERROR;

    LocalDriverDir = (char*)pTemp;
    LocalDriverDir = LocalDriverDir + "\\";

    i = -1;
    while (SourceFiles->Count > ++i)
    {
        DestFileName = LocalDriverDir +
            ExtractFileName(SourceFiles->Strings[i]);

        ::CopyFile(SourceFiles->Strings[i].c_str(), DestFileName.c_str(), NULL);

        ReturnStrings->Add(DestFileName);
    }

    free(pTemp);

    return ReturnStrings;
}

bool TPrinterControl::ValidateMonitor(AnsiString MonitorName)
{
    MONITOR_INFO_2 *pLocalMonitors = new MONITOR_INFO_2;
    DWORD dwSize;
    DWORD dwBytesNeeded;

```

FIG. 6.41

```

DWORD dwReturned;
int i;

if (0 == MonitorName.AnsiCompareIC("Client Printer Port"))
{
    return true;
}

//Get the memory needed.
EnumMonitors(NULL, 2, NULL, 0, &dwSize, &dwReturned);

pLocalMonitors = (MONITOR_INFO_2*)malloc(dwSize);

if (EnumMonitors(NULL, 2, (unsigned char*)pLocalMonitors, dwSize, &dwBytesNeeded,
    &dwReturned))
{
    i = -1;
    while ((int)dwReturned > ++i)
    {
        if (0 == strcmp(MonitorName.c_str(), pLocalMonitors[i].pName))
            break;
    }
}

if (i >= (int)dwReturned || 0 >= dwReturned)
{
    free(pLocalMonitors);
    return false;
}

free(pLocalMonitors);
return true;
}

bool TPrinterControl::ValidatePort(AnsiString PortName, AnsiString PortMonitor)
{
    HINSTANCE hLib;
    PORT_INFO_1 *pLocalPorts = new PORT_INFO_1;
    PORT_INFO_1 PortInfo;
    DWORD dwSize;
    DWORD dwReturned;
    DWORD dwBytesNeeded;
    int i;

    EnumPorts(NULL, 1, (unsigned char*)pLocalPorts, 0, &dwSize, &dwReturned);

    pLocalPorts = (PORT_INFO_1*)malloc(dwSize);

    EnumPorts(NULL, 1, (unsigned char*)pLocalPorts, dwSize, &dwBytesNeeded, &dwReturned);

    i = -1;
    while ((int)dwReturned > ++i)
    {
        if (0 == strcmp(PortName.c_str(), pLocalPorts[i].pName))
            break;
    }

    free(pLocalPorts);

```

FIG. 6.42

```

//We found the port.
if ((int)dwReturned > i)
    return true;

hLib = LoadLibrary("winspool.drv");

if (NULL == hLib)
    return false;

ADDPOTEX pfnAddPortEx = (ADDPOTEX)GetProcAddress(hLib, "AddPortExA");

PortInfo.pName = PortName.c_str();

if (pfnAddPortEx)
{
    if (!(*pfnAddPortEx)(NULL, 1, (unsigned char*)&PortInfo,
        (WCHAR*)PortMonitorDescription.c_str()))
    {
        FreeLibrary(hLib);
        return false;
    }
}

FreeLibrary(hLib);
return true;
}

bool TPrinterControl::ValidateDriver(AnsiString DriverName)
{
    DRIVER_INFO_3 *pRemoteDriver;
    DRIVER_INFO_3 NewLocalDriverInfo;
    TStringList *LocalDrivers = new TStringList;
    TStringList *DriverFilesToCopy = new TStringList;
    TStringList *CopiedDriverFiles = new TStringList;
    int i;
    int j;
    int nPos;
    int NullTerminatorsFound;
    BYTE *pTemp;
    DWORD dwBufferSize = 1024;

    LocalDrivers = GetLocalDrivers();

    i = -1;
    while (LocalDrivers->Count > ++i)
    {
        if (0 == strcmp(LocalDrivers->Strings[i].c_str(), DriverName.c_str()))
        {
            LocalDrivers->Free();
            return true;
        }
    }

    pRemoteDriver = GetRemoteDriverInfo(SourceServerName, DriverName);

    if (NULL == pRemoteDriver)
        return false;

```

FIG. 6.43


```

DriverFilesToCopy->Add(pRemoteDriver->pDriverPath);
DriverFilesToCopy->Add(pRemoteDriver->pDataFile);
DriverFilesToCopy->Add(pRemoteDriver->pConfigFile);
DriverFilesToCopy->Add(pRemoteDriver->pHelpFile);

i = -1;
j = -1;
NullTerminatorsFound = 0;
pTemp = (BYTE*)malloc(dwBufferSize);
ZeroMemory(pTemp, dwBufferSize);
while (++i < (int)dwBufferSize && 2 > NullTerminatorsFound)
{
    if ('\0' == pRemoteDriver->pDependentFiles[i])
    {
        DriverFilesToCopy->Add((char*)pTemp);
        ZeroMemory(pTemp, dwBufferSize);

        j = -1;
        NullTerminatorsFound++;

        continue;
    }

    pTemp[++j] = pRemoteDriver->pDependentFiles[i];
}

CopiedDriverFiles = CopyDriverFiles(DriverFilesToCopy);

NewLocalDriverInfo.cVersion = pRemoteDriver->cVersion;
NewLocalDriverInfo.pName = pRemoteDriver->pName;
NewLocalDriverInfo.pEnvironment = pRemoteDriver->pEnvironment;
NewLocalDriverInfo.pMonitorName = pRemoteDriver->pMonitorName;
NewLocalDriverInfo.pDefaultDataType = pRemoteDriver->pDefaultDataType;

i = -1;
NewLocalDriverInfo.pDriverPath = CopiedDriverFiles->Strings[++i].c_str();
NewLocalDriverInfo.pDataFile = CopiedDriverFiles->Strings[++i].c_str();
NewLocalDriverInfo.pConfigFile = CopiedDriverFiles->Strings[++i].c_str();
NewLocalDriverInfo.pHelpFile = CopiedDriverFiles->Strings[++i].c_str();

NewLocalDriverInfo.pDependentFiles = (char*)malloc(dwBufferSize);
ZeroMemory(NewLocalDriverInfo.pDependentFiles, dwBufferSize);
nPos = -1;
while (CopiedDriverFiles->Count > ++i)
{
    j = 0;
    while(CopiedDriverFiles->Strings[i].Length() >= ++j)
        NewLocalDriverInfo.pDependentFiles[++nPos] = CopiedDriverFiles->Strings[i][j];

    NewLocalDriverInfo.pDependentFiles[++nPos] = '\0';
}
NewLocalDriverInfo.pDependentFiles[++nPos] = '\0';

if (!AddPrinterDriver(NULL, 3, (unsigned char*)&NewLocalDriverInfo))
{
    delete pRemoteDriver;
}

```

FIG. 6.44

```

    pRemoteDriver = NULL;
    LocalDrivers->Free();
    return false;
}

delete pRemoteDriver;
pRemoteDriver = NULL;
LocalDrivers->Free();

return true;
}

bool TPrinterControl::PrinterSetOwnerOnlyRights(AnsiString PrinterName)
{
    HANDLE                                hPrinter = NULL;
    PRINTER_DEFAULTS                      pdPrinter;
    LPPRINTER_INFO_3                     pPrinterInfo = NULL;
    PACCESS_ALLOWED_ACE                  pTempAce;
    PSID                                  psidOwner;
    PACL                                  pPrinterNewACL;
    DWORD                                 dwBytesNeeded;
    BOOL                                  bOwnerDefaulted;

    // Assign desired access level to PRINTER_DEFAULTS
    pdPrinter.DesiredAccess = PRINTER_ALL_ACCESS;
    pdPrinter.pDevMode = NULL;
    pdPrinter.pDatatype = NULL;

    //Open the printer and add the User
    if (0 != OpenPrinter(PrinterName.c_str(),&hPrinter,&pdPrinter))
    {
        //Get the required value of dwBytesNeeded. And allocate the memory for pPrinterInfo.
        GetPrinter(hPrinter,3,(LPBYTE)pPrinterInfo,0,&dwBytesNeeded);
        pPrinterInfo = (LPPRINTER_INFO_3)malloc(dwBytesNeeded);

        //Get the actual printer stuff and add the ACE to the DACL.
        if (0 != GetPrinter(hPrinter,3,(LPBYTE)pPrinterInfo,dwBytesNeeded,&dwBytesNeeded))
        {
            if (GetSecurityDescriptorOwner(pPrinterInfo->
                pSecurityDescriptor,&psidOwner,&bOwnerDefaulted))
            {
                //Multiply by 2 to get the size needed for 2 ACEs.
                DWORD dwSize = sizeof(ACL) + 2*(sizeof(ACCESS_ALLOWED_ACE) +
                    GetLengthSid(psidOwner) - sizeof(DWORD));

                pPrinterNewACL = (PACL)malloc(dwSize);
                InitializeAcl(pPrinterNewACL, dwSize, ACL_REVISION);

                pTempAce = (PACCESS_ALLOWED_ACE)malloc(sizeof(ACCESS_ALLOWED_ACE));

                //For some reason, there are 2 ACEs for "Full Control".Add the ACEs.
                AddAccessAllowedAce(pPrinterNewACL,ACL_REVISION,GENERIC_ALL,psidOwner);
                if (0 != GetAce(pPrinterNewACL,pPrinterNewACL->AceCount -
                    1,(LPVOID*)&pTempAce))

                    pTempAce->Header.AceFlags = OBJECT_INHERIT_ACE |

```

FIG. 6.45

INHERIT_ONLY_ACE;

```
AddAccessAllowedAce(pPrinterNewACL,ACL_REVISION,PRINTER_ALL_ACCESS,psidOwner);
    if (0 != GetAce(pPrinterNewACL,pPrinterNewACL->AceCount -,(LPVOID*)&pTempAce))
        pTempAce->Header.AceFlags = CONTAINER_INHERIT_ACE;

    InitializeSecurityDescriptor(pPrinterInfo->pSecurityDescriptor,
        SECURITY_DESCRIPTOR_REVISION);

    SetSecurityDescriptorDacl(pPrinterInfo->pSecurityDescriptor,TRUE,
        pPrinterNewACL,FALSE);

    SetPrinter(hPrinter,3,(LPBYTE)pPrinterInfo,0);
}
}

free(pPrinterInfo);
}

else
    return false;

//Close the printer.
ClosePrinter(hPrinter);

return true;
```

```
bool TPrinterControl::PrinterSetCurrentUserOnlyRights(AnsiString PrinterName)
```

```
{
    HANDLE                hPrinter = NULL;
    PRINTER_DEFAULTS      pdPrinter;
    LPPRINTER_INFO_3      pPrinterInfo = NULL;
    PACCESS_ALLOWED_ACE   pTempAce;
    PSID                  psidOwner;
    PSID                  psidCurrentUser;
    PACL                  pPrinterNewACL;
    DWORD                 dwBytesNeeded = 0;
    DWORD                 dwSizeDomain = 256;
    BOOL                  bOwnerDefaulted;
    char                  szUserName[256];
    char                  szDomainController[256];
    char                  szDomainName[256];
    PSID_NAME_USE          peUse;

    // Assign desired access level to PRINTER_DEAFULTS
    pdPrinter.DesiredAccess = PRINTER_ALL_ACCESS;
    pdPrinter.pDevMode = NULL;
    pdPrinter.pDatatype = NULL;

    //Open the printer and add the User
    if (0 != OpenPrinter(PrinterName.c_str(),&hPrinter,&pdPrinter))
    {
        //Get the required value of dwBytesNeeded. And allocate the memory for pPrinterInfo.
        GetPrinter(hPrinter,3,(LPBYTE)pPrinterInfo,0,&dwBytesNeeded);
        pPrinterInfo = (LPPRINTER_INFO_3)malloc(dwBytesNeeded);
    }
}
```

FIG. 6.46

```

//Get the actual printer stuff and add the ACE to the DACL.
if (0 != GetPrinter(hPrinter,3,(LPBYTE)pPrinterInfo,dwBytesNeeded,&dwBytesNeeded))
{
strcpy(szDomainController, getenv("LOGONSERVER"));
strcpy(szUserName, getenv("USERNAME"));
strcpy(szDomainName, getenv("USERDOMAIN"));

dwBytesNeeded = 0;
dwSizeDomain = 256;
LookupAccountName(szDomainController, szUserName, psidCurrentUser,
    &dwBytesNeeded, szDomainName, &dwSizeDomain, peUse);

peUse = (PSID_NAME_USE)malloc(sizeof(SID_NAME_USE));
psidCurrentUser = (PSID)malloc(dwBytesNeeded);

if (LookupAccountName(szDomainController, szUserName, psidCurrentUser,
    &dwBytesNeeded, szDomainName, &dwSizeDomain, peUse))
    {
        //Multiply by 2 to get the size needed for 2 ACEs.
        DWORD dwSize = sizeof(ACL) + 2*(sizeof(ACCESS_ALLOWED_ACE) +
            GetLengthSid(psidCurrentUser) - sizeof(DWORD));

        pPrinterNewACL = (PACL)malloc(dwSize);
        InitializeAcl(pPrinterNewACL, dwSize, ACL_REVISION);

        pTempAce = (PACCESS_ALLOWED_ACE)malloc(sizeof(ACCESS_ALLOWED_ACE));

        //For some reason, there are 2 ACEs for "Full Control".Add the ACEs.
        AddAccessAllowedAce(pPrinterNewACL,ACL_REVISION,GENERIC_ALL,psidCurrentUser);
        if (0 != GetAce(pPrinterNewACL,pPrinterNewACL->AceCount -
            1,(LPVOID*)&pTempAce))
            pTempAce->Header.AceFlags = OBJECT_INHERIT_ACE |
                INHERIT_ONLY_ACE;

        AddAccessAllowedAce(pPrinterNewACL,ACL_REVISION,PRINTER_ALL_ACCESS,psidCurrentUser);
        if (0 != GetAce(pPrinterNewACL,pPrinterNewACL->AceCount -
            1,(LPVOID*)&pTempAce))
            pTempAce->Header.AceFlags = CONTAINER_INHERIT_ACE;

        InitializeSecurityDescriptor(pPrinterInfo->pSecurityDescriptor,
            SECURITY_DESCRIPTOR_REVISION);

        SetSecurityDescriptorDacl(pPrinterInfo->pSecurityDescriptor,TRUE,
            pPrinterNewACL,FALSE);

        SetPrinter(hPrinter,3,(LPBYTE)pPrinterInfo,0);
    }
}

free(pPrinterInfo);
}
else
return false;

//Close the printer.

```

FIG. 6.47

```

        ClosePrinter(hPrinter);

        return true;
    }

bool TPrinterControl::PrinterAddAccessRights(AnsiString PrinterName, TStringList *Users, int nAccess)
{
    int i = -1;

    while (Users->Count > ++i)
    {
        PrinterAddAccessRights(PrinterName, Users->Strings[i], nAccess);
    }

    return true;
}

bool TPrinterControl::PrinterAddAccessRights(AnsiString PrinterName, AnsiString UserName, int nAccess)
{
    ACL_SIZE_INFORMATION    ACLInformation;
    PRINTER_DEFAULTS pdPrinter;
    LPPRINTER_INFO_3 pPrinterInfo = NULL;
    PACCESS_ALLOWED_ACE pTempAce;
    HANDLE hPrinter = NULL;
    PACL pPrinterACL;
    PACL pPrinterNewACL;
    DWORD        dwBytesNeeded;
    BOOL bDaclPresent = FALSE;
    BOOL bDaclDefaulted = FALSE;
    int i;

    //Used for LookupAccountName().
    PSID psidUserName;
    PSID_NAME_USE peUse;
    char szDomainName[256];
    DWORD        dwSizeDomain = 256;

    // Assign desired access level to PRINTER_DEFAULTS
    pdPrinter.DesiredAccess = PRINTER_ALL_ACCESS;
    pdPrinter.pDevMode = NULL;
    pdPrinter.pDatatype = NULL;

    //Let's get the SID of the user.
    dwSizeDomain = 256;
    dwBytesNeeded = 0;

    LookupAccountName(NULL, UserName.c_str(), psidUserName, &dwBytesNeeded,
        szDomainName, &dwSizeDomain, peUse);

    peUse = (PSID_NAME_USE)malloc(sizeof(SID_NAME_USE));
    psidUserName = (PSID)malloc(dwBytesNeeded);

    if (0 == LookupAccountName(NULL, UserName.c_str(), psidUserName, &dwBytesNeeded, szDomainName,
        &dwSizeDomain, peUse))
    {

```

FIG. 6.48

```

free(peUse);
free(psidUserName);
    return false;
}

```

```

//Open the printer and add the User
if (0 != OpenPrinter(PrinterName.c_str(),&hPrinter,&pdPrinter))
{

    //Get the required value of dwBytesNeeded. And allocate the memory for pPrinterInfo.
    GetPrinter(hPrinter,3,(LPBYTE)pPrinterInfo,0,&dwBytesNeeded);
    pPrinterInfo = (LPPRINTER_INFO_3)malloc(dwBytesNeeded);

    //Get the actual printer stuff and add the ACE to the DACL.
    if (0 != GetPrinter(hPrinter,3,(LPBYTE)pPrinterInfo,dwBytesNeeded,&dwBytesNeeded))
    {

        // Get printer ACL
        GetSecurityDescriptorDacl(pPrinterInfo->pSecurityDescriptor,&bDaclPresent,
                                &pPrinterACL,&bDaclDefaulted);

        // Get the number of entries in the ACL
        GetAclInformation(pPrinterACL,&ACLInformation,sizeof(ACLInformation),
                        AclSizeInformation);

        //Multiply by 2 to get the size needed for 2 ACEs.
        DWORD dwSize = pPrinterACL->AclSize + 2*(sizeof(ACCESS_ALLOWED_ACE) +
                                GetLengthSid(psidUserName) - sizeof(DWORD));

        pPrinterNewACL = (PACL)malloc(dwSize);
        InitializeAcl(pPrinterNewACL, dwSize, ACL_REVISION);

        //Copy the old ACL's ACEs to the new ACL.
        pTempAce = (PACCESS_ALLOWED_ACE)malloc(sizeof(ACCESS_ALLOWED_ACE));
        i = -1;
        while (pPrinterACL->AceCount > ++i)
        {
            if (0 != GetAce(pPrinterACL, i,(LPVOID*)&pTempAce))
                AddAce(pPrinterNewACL, ACL_REVISION, MAXDWORD, pTempAce,
pTempAce->Header.AceSize);
        }

        switch(nAccess)
        {
            case(CONTROL_FULL):
                //For some reason, there are 2 ACEs for "Full Control".Add the ACEs.
                AddAccessAllowedAce(pPrinterNewACL,ACL_REVISION,GENERIC_ALL,psidUserName);
                if (0 != GetAce(pPrinterNewACL,pPrinterNewACL->AceCount -
1,(LPVOID*)&pTempAce))
                    pTempAce->Header.AceFlags = OBJECT_INHERIT_ACE |
INHERIT_ONLY_ACE;

                AddAccessAllowedAce(pPrinterNewACL,ACL_REVISION,PRINTER_ALL_ACCESS,psidUserName);
                if (0 != GetAce(pPrinterNewACL,pPrinterNewACL->AceCount -

```

FIG. 6.49

```

1,(LPVOID*)&pTempAce))
                                pTempAce->Header.AceFlags = CONTAINER_INHERIT_ACE;
                                break;

                                default:
                                    break;
                                }

                                InitializeSecurityDescriptor(pPrinterInfo-
>pSecurityDescriptor,SECURITY_DESCRIPTOR_REVISION);
                                SetSecurityDescriptorDacl(pPrinterInfo->pSecurityDescriptor,TRUE,pPrinterNewACL,FALSE);

                                SetPrinter(hPrinter,3,(LPBYTE)pPrinterInfo,0);
                                }

                                free(pPrinterInfo);
                                free(peUse);
                                }
else
    return false;

    //Close the printer.
    ClosePrinter(hPrinter);

    return true;
}

bool TPrinterControl::RemapPort(AnsiString Port, AnsiString Monitor)
{
    if (Port.IsEmpty() || Monitor.IsEmpty())
    {
        Messages->Add("Unable to remap Port!");
        return false;
    }

    NewPortName = Port;
    NewPortMonitor = Monitor;

    return true;
}

TStringList *TPrinterControl::GetConfigFileList()
{
    TStringList *ConfigFiles = new TStringList;
    TStringList *Filenames = new TStringList;
    int i;

    EnumerateFiles(PrtInfoPath, Filenames, false, NULL);

    i = -1;
    while (Filenames->Count > ++i)
    {
        Filenames->Strings[i] = JustFilenameL(Filenames->Strings[i]);

        //Check for dots.
        if (0 == Filenames->Strings[i].AnsiCompareIC(".") ||
            0 == Filenames->Strings[i].AnsiCompareIC(".."))

```

FIG. 6.50

```

    {
        continue;
    }

    Filenames->Strings[i] = Filenames->Strings[i].SubString(
        1, (Filenames->Strings[i].Length() - 4));

    if (0 > ConfigFiles->IndexOf(Filenames->Strings[i]) &&
        !Filenames->Strings[i].IsEmpty())
    {
        ConfigFiles->Add(Filenames->Strings[i]);
    }
}

Filenames->Free();

return ConfigFiles;
}

TStringList *TPrinterControl::LoadPrinterInfoFromFile(AnsiString PrinterName)
{
    TStringList *PrinterInfo = new TStringList;
    AnsiString ReturnedPrinterName;
    AnsiString ReturnedPortName;
    AnsiString ReturnedPortMonitorName;

    if (!ReadPrinterInfo(PrtInfoPath + "\\\" + PrinterName + ".Prt"))
    {
        Messages->Add("Error reading PrinterInfo from file!");
    }

    ReturnedPrinterName = SelectedPrinterInfo->pPrinterName;
    ReturnedPortName = SelectedPrinterInfo->pPortName;
    ReturnedPortMonitorName = GetPortMonitor(SelectedPrinterInfo->pPortName);

    PrinterInfo->Add(ReturnedPrinterName);
    PrinterInfo->Add(ReturnedPortName);
    PrinterInfo->Add(ReturnedPortMonitorName);

    return PrinterInfo;
}

bool TPrinterControl::PrinterPropertiesDialog(AnsiString PrinterName, HANDLE hWnd)
{
    HANDLE hPrinter;
    DWORD dwNeeded, dwReturned;
    PRINTER_INFO_2* pPrtInfo;
    PRINTER_DEFAULTS pdPrinter;

    // Assign desired access level to PRINTER_DEFAULTS
    pdPrinter.DesiredAccess = PRINTER_ALL_ACCESS;

    pdPrinter.pDevMode = NULL;
    pdPrinter.pDatatype = NULL;

    //Open handle to printer.
    if(!OpenPrinter(PrinterName.c_str(), &hPrinter, &pdPrinter))

```

FIG. 6.51


```

{
    Messages->Add("OpenPrinter() Failed!");
    return false;
}

//Select the default printer.
if(NULL!=hPrinter){

    // Get the buffer size needed
    GetPrinter(hPrinter,2,NULL,0,&dwNeeded);

    pPrtInfo=(PRINTER_INFO_2*)malloc(dwNeeded);
    ZeroMemory(pPrtInfo, dwNeeded);

    //get the printer info
    GetPrinter(hPrinter,2,(unsigned char*)pPrtInfo,dwNeeded,&dwReturned);

    if (!PrinterProperties(hWnd, hPrinter))
    {
        Messages->Add("PrinterProperties() Failed!");
        ClosePrinter(hPrinter);
        free(pPrtInfo);
        return false;
    }

    //Close the handle to the printer.
    ClosePrinter(hPrinter);
}

free(pPrtInfo);
return true;
}

bool TPrinterControl::DeleteLocalPrinter(AnsiString PrinterName)
{
    HANDLE hPrinter;
    PRINTER_DEFAULTS pdPrinter;

    // Assign desired access level to PRINTER_DEAFULTS
    pdPrinter.DesiredAccess = PRINTER_ALL_ACCESS;
    pdPrinter.pDevMode = NULL;
    pdPrinter.pDatatype = NULL;

    //Open handle to printer.
    if(!OpenPrinter(PrinterName.c_str(), &hPrinter, &pdPrinter))
    {
        Messages->Add("DeletePrinter() Failed!");
        return false;
    }

    //Select the default printer.
    if(NULL == hPrinter)
    {
        Messages->Add("DeletePrinter() Failed! NULL Handle.");
        return false;
    }
}

```

FIG. 6.52

```

SetPrinter(hPrinter, 0, NULL, PRINTER_CONTROL_PURGE);

Sleep(250);

DeletePrinter(hPrinter);

    //Close the handle to the printer.
    ClosePrinter(hPrinter);

return true;
}

bool TPrinterControl::DeletePrinterConfig(AnsiString PrinterConfigName)
{
    AnsiString PrinterConfigPath;
    bool bReturn = true;

    PrinterConfigPath = PrtInfoPath + "\\\" + PrinterConfigName;

    if (FileExists(PrinterConfigPath + ".Prt") &&
        FileExists(PrinterConfigPath + ".Dev"))
    {
        if (!DeleteFile(PrinterConfigPath + ".Prt") ||
            !DeleteFile(PrinterConfigPath + ".Dev"))
        {
            bReturn = false;
        }
    }
    else
    {
        Messages->Add("Files Not Found: " + PrinterConfigPath);
        bReturn = false;
    }

    return bReturn;
}

AnsiString TPrinterControl::GetPrinterShareName(AnsiString PrinterName)
{
    HANDLE hPrinter;
    DWORD dwNeeded, dwReturned;
    PRINTER_INFO_2* pPrtInfo;
    PRINTER_DEFAULTS pdPrinter;
    AnsiString ShareName;
    AnsiString ServerName;
    AnsiString FullShareName;

    // Assign desired access level to PRINTER_DEFAULTS
    pdPrinter.DesiredAccess = PRINTER_ACCESS_USE;
    pdPrinter.pDevMode = NULL;
    pdPrinter.pDatatype = NULL;

    //Open handle to printer.
    if(!OpenPrinter(PrinterName.c_str(), &hPrinter, &pdPrinter))
    {
        Messages->Add("OpenPrinter() Failed!");
    }
}

```

FIG. 6.53

```

    return "";
}

//Select the default printer.
if(NULL!=hPrinter){

    // Get the buffer size needed
    GetPrinter(hPrinter,2,NULL,0,&dwNeeded);

    pPrtInfo=(PRINTER_INFO_2*)malloc(dwNeeded);
    ZeroMemory(pPrtInfo, dwNeeded);

    //get the printer info
    GetPrinter(hPrinter,2,(unsigned char*)pPrtInfo,dwNeeded,&dwReturned);

    ShareName = pPrtInfo->pShareName;
    ServerName = pPrtInfo->pServerName;

    //Close the handle to the printer.
    ClosePrinter(hPrinter);
}

free(pPrtInfo);

if (ServerName.IsEmpty())
    FullShareName = ShareName;
else
    FullShareName = ServerName + "\\\" + ShareName;

return FullShareName;
}

AnsiString TPrinterControl::GetPrinterFullName(AnsiString PrinterName)
{
    HANDLE hPrinter;
    DWORD dwNeeded, dwReturned;
    PRINTER_INFO_2* pPrtInfo;
    PRINTER_DEFAULTS pdPrinter;
    AnsiString FullName;

    // Assign desired access level to PRINTER_DEAFULTS
    pdPrinter.DesiredAccess = PRINTER_ACCESS_USE;
    pdPrinter.pDevMode = NULL;
    pdPrinter.pDatatype = NULL;

    //Open handle to printer.
    if(!OpenPrinter(PrinterName.c_str(), &hPrinter, &pdPrinter))
    {
        Messages->Add("OpenPrinter() Failed!");
        return "";
    }

    //Select the default printer.
    if(NULL!=hPrinter){

```

FIG. 6.54

```

        // Get the buffer size needed
        GetPrinter(hPrinter,2,NULL,0,&dwNeeded);

        pPrtInfo=(PRINTER_INFO_2*)malloc(dwNeeded);
        ZeroMemory(pPrtInfo, dwNeeded);

        //get the printer info
        GetPrinter(hPrinter,2,(unsigned char*)pPrtInfo,dwNeeded,&dwReturned);

        FullName = pPrtInfo->pPrinterName;

        //Close the handle to the printer.
        ClosePrinter(hPrinter);
    }

    free(pPrtInfo);

    return FullName;
}

bool TPrinterControl::ClearNetworkPrinters()
{
    DWORD dwBytesNeeded;
    DWORD dwPrtRet;
    LPPRINTER_INFO_4 pPrtInfo;
    int i=0;

    //Get the memory needed for structure.
    EnumPrinters(PRINTER_ENUM_CONNECTIONS,NULL,4,NULL,0,&dwBytesNeeded,&dwPrtRet);

    //Allocate the memory for the structure.
    pPrtInfo =(LPPRINTER_INFO_4)malloc(dwBytesNeeded);

    //Enumerate the printers.
    if
    (!EnumPrinters(PRINTER_ENUM_CONNECTIONS,NULL,4,(LPBYTE)pPrtInfo,dwBytesNeeded,&dwBytesNeeded,&dwPrtR
    et))
        return false;

    //Delete the printer connection.
    for (i = 0; i < (int)dwPrtRet; i++)
        DeletePrinterConnection((pPrtInfo++)->pPrinterName);

    return true;
}

bool TPrinterControl::SetIcaPrinterRights()
{
    TStringList *LocalPrinterList = new TStringList;
    PRINTER_INFO_2 *InstalledPrinterInfo = new PRINTER_INFO_2;
    DWORD InstalledPrinterInfoReturned;
    DWORD dwSize;
    DWORD dwNeeded;
    AnsiString Comment;
    AnsiString PrinterName;

```

FIG. 6.55

```

int i;

EnumPrinters(PRINTER_ENUM_LOCAL, NULL, 2,(BYTE*)InstalledPrinterInfo,
0, &dwSize, &InstalledPrinterInfoReturned);

InstalledPrinterInfo = (PRINTER_INFO_2*)malloc(dwSize);
ZeroMemory(InstalledPrinterInfo, dwSize);

if (!EnumPrinters(PRINTER_ENUM_LOCAL, NULL, 2,(BYTE*)InstalledPrinterInfo,
dwSize, &dwNeeded, &InstalledPrinterInfoReturned))
{
    return false;
}

i = -1;
while ((int)InstalledPrinterInfoReturned > ++i)
{
    PrinterName = InstalledPrinterInfo[i].pPrinterName;
    Comment = InstalledPrinterInfo[i].pComment;
    if (0 < Comment.AnsiPos("Auto Created Client Printer"))
    {
        PrinterSetOwnerOnlyRights(PrinterName);
        PrinterAddAccessRights(PrinterName, "SYSTEM", CONTROL_FULL);
    }
}

free(InstalledPrinterInfo);
return true;
}

bool TPrinterControl::CopyConfiguration(AnsiString Source, AnsiString Destination)
{
    AnsiString PrinterConfigSourcePath;
    AnsiString PrinterConfigDestPath;

    PrinterConfigSourcePath = PrtInfoPath + "\\\" + Source;
    PrinterConfigDestPath = PrtInfoPath + "\\\" + Destination;

    if (FileExists(PrinterConfigSourcePath + ".Prt") &&
        FileExists(PrinterConfigSourcePath + ".Dev"))
    {
        if (0 == ::CopyFile(String(PrinterConfigSourcePath + ".Prt").c_str(),
            String(PrinterConfigDestPath + ".Prt").c_str(), NULL))
        {
            return false;
        }
        if (0 == ::CopyFile(String(PrinterConfigSourcePath + ".Dev").c_str(),
            String(PrinterConfigDestPath + ".Dev").c_str(), NULL))
        {
            DeleteFile(PrinterConfigDestPath + ".Prt");
            return false;
        }
    }
    else
    {
        Messages->Add("Files Not Found: " + PrinterConfigSourcePath);
        return false;
    }
}

```

FIG. 6.56

```

    }

    return true;
}

bool TPrinterControl::SaveLocalDriver(AnsiString DriverName)
{
    AnsiString PrinterName;
    HANDLE hPrinter;

    PrinterName = "PMPAdmin#" + DriverName;
    SelectedPrinterInfo->pPrinterName = PrinterName.c_str();
    SelectedPrinterInfo->pPortName = "LPT1:";
    SelectedPrinterInfo->pDriverName = DriverName.c_str();
    SelectedPrinterInfo->pPrintProcessor = "winprint";

    //Add the printer
    hPrinter = AddPrinter(NULL, 2, (unsigned char*)SelectedPrinterInfo);

    if (NULL == hPrinter)
        return false;

    ClosePrinter(hPrinter);
    hPrinter = NULL;

    if (!SaveLocalPrinter(PrinterName, DriverName))
    {
        DeleteLocalPrinter(PrinterName);
        return false;
    }

    DeleteLocalPrinter(PrinterName);
    return true;
}

AnsiString TPrinterControl::CleanupFilename(AnsiString Filename)
{
    int Index;
    int i;
    TStringList *InvalidList = new TStringList;

    if (Filename.IsEmpty())
        return Filename;

    InvalidList->Add("\\");
    InvalidList->Add("/");
    InvalidList->Add(":");
    InvalidList->Add("?");
    InvalidList->Add("*");

    i = -1;
    while (InvalidList->Count > ++i)
    {
        Index = Filename.AnsiPos(InvalidList->Strings[i]);
        if (0 < Index)
        {

```

FIG. 6.57

```

        Filename.Delete(Index, 1);
        Filename = CleanupFilename(Filename);
    }
}

return Filename;
}

AnsiString TPrinterControl::GetIcaClientPort(AnsiString OldPort)
{
    int BackSlash = 0;
    AnsiString NewPort;
    AnsiString Port;

    BackSlash = OldPort.AnsiPos("\\");

    Port = OldPort.SubString( (BackSlash + 1),
        (OldPort.Length() - BackSlash) );

    NewPort = "Client\\" + String(getenv("CLIENTNAME")) + "\\#\" + Port;

    return NewPort;
}

PRINTER_INFO_2 *TPrinterControl::GetPrinterInfo2(AnsiString PrinterName)
{
    HANDLE hPrinter;
    DWORD dwNeeded, dwReturned;
    PRINTER_INFO_2* pPrtInfo;
    PRINTER_DEFAULTS pdPrinter;

    // Assign desired access level to PRINTER_DEFAULTS
    pdPrinter.DesiredAccess = PRINTER_ACCESS_USE;
    pdPrinter.pDevMode = NULL;
    pdPrinter.pDatatype = NULL;

    //Open handle to printer.
    if(!OpenPrinter(PrinterName.c_str(), &hPrinter, &pdPrinter))
    {
        return NULL;
    }

    //Select the default printer.
    if(NULL!=hPrinter){

        // Get the buffer size needed
        GetPrinter(hPrinter,2,NULL,0,&dwNeeded);

        pPrtInfo=(PRINTER_INFO_2*)malloc(dwNeeded);
        ZeroMemory(pPrtInfo, dwNeeded);

        //get the printer info
        GetPrinter(hPrinter,2,(unsigned char*)pPrtInfo,dwNeeded,&dwReturned);

        //Close the handle to the printer.
        ClosePrinter(hPrinter);
    }
}

```

FIG. 6.58

```

    }

    return pPrtInfo;
}

AnsiString TPrinterControl::GetStatusString(DWORD dwStatus)
{
    AnsiString Status;

    switch(dwStatus)
    {
        case(PRINTER_STATUS_BUSY):
            Status = "Busy";
            break;
        case(PRINTER_STATUS_DOOR_OPEN):
            Status = "Door Open";
            break;
        case(PRINTER_STATUS_ERROR):
            Status = "Error";
            break;
        case(PRINTER_STATUS_INITIALIZING):
            Status = "Initializing";
            break;
        case(PRINTER_STATUS_IO_ACTIVE):
            Status = "I/O Active";
            break;
        case(PRINTER_STATUS_MANUAL_FEED):
            Status = "Manual Feed";
            break;
        case(PRINTER_STATUS_NO_TONER):
            Status = "No Toner";
            break;
        case(PRINTER_STATUS_NOT_AVAILABLE):
            Status = "Not Available";
            break;
        case(PRINTER_STATUS_OFFLINE):
            Status = "Offline";
            break;
        case(PRINTER_STATUS_OUT_OF_MEMORY):
            Status = "Out of Memory";
            break;
        case(PRINTER_STATUS_OUTPUT_BIN_FULL):
            Status = "Output Bin Full";
            break;
        case(PRINTER_STATUS_PAGE_PUNT):
            Status = "Page Punt";
            break;
        case(PRINTER_STATUS_PAPER_JAM):
            Status = "Paper Jam";
            break;
        case(PRINTER_STATUS_PAPER_OUT):
            Status = "Paper Out";
            break;
        case(PRINTER_STATUS_PAPER_PROBLEM):
            Status = "Paper Problem";
            break;
        case(PRINTER_STATUS_PAUSED):

```

FIG. 6.59


```

        Status = "Paused";
        break;
    case(PRINTER_STATUS_PENDING_DELETION):
        Status = "Pending Deletion";
        break;
    case(PRINTER_STATUS_POWER_SAVE):
        Status = "Power Save";
        break;
    case(PRINTER_STATUS_PRINTING):
        Status = "Printing";
        break;
    case(PRINTER_STATUS_PROCESSING):
        Status = "Processing";
        break;
    case(PRINTER_STATUS_SERVER_UNKNOWN):
        Status = "Server Unknown";
        break;
    case(PRINTER_STATUS_TONER_LOW):
        Status = "Toner Low";
        break;
    case(PRINTER_STATUS_USER_INTERVENTION):
        Status = "User Intervention";
        break;
    case(PRINTER_STATUS_WAITING):
        Status = "Waiting";
        break;
    case(PRINTER_STATUS_WARMING_UP):
        Status = "Warming Up";
        break;
    default:
        Status = "Ready";
        break;
}

return Status;

```

FIG. 6.60